Eesti Maaülikool
Tartu Tehnikakolledž

Jan Bogdanov

NUTITELEFONI JUHTMEVABA LISASEADE VALGUSTATUSE NING VALGUSE VÄRVUSTEMPERATUURI MÕÕTMISEKS

WIRELESS ACCESSORY FOR A SMARTPHONE TO MEASURE ILLUMINANCE AND COLOUR TEMPERATURE

Rakenduskõrgharidusõppe lõputöö

Tehnotroonika õppekava

Juhendajad:
Tõnu Leemet, PhD
Tanel Ainla, MSc

Tartu 2017

LÜHIKOKKUVÕTE

<table>
<tr><td colspan="2">Eesti Maaülikool
Kreutzwaldi 1, Tartu 51014</td><td colspan="2">Rakenduskõrgharidusõppe lõputöö
lühikokkuvõte</td></tr>
<tr><td colspan="2">Autor: Jan Bogdanov</td><td colspan="2">Õppekava: Tehnotroonika</td></tr>
<tr><td colspan="4">Pealkiri: Nutitelefoni juhtmevaba lisaseade valgustatuse ning valguse värvitemperatuuri mõõtmiseks</td></tr>
<tr><td>Lehekülgi: 35</td><td>Jooniseid: 17</td><td>Tabeleid: 2</td><td>Lisasid: 3</td></tr>
<tr><td colspan="4">Osakond: Tartu Tehnikakolledž
Uurimisvaldkond: Projekt
Juhendaja(d): Tõnu Leemet, Tanel Ainla
Kaitsmiskoht ja -aasta: Tartu, 2017</td></tr>
<tr><td colspan="4">Meid ümbritseva valguse omaduste mõõtmine on oluline mitmete elukutsete esindajatele. Tänapäeval valdavalt kasutuses olevatel mõõteseadmetel on teatud piirangud, mida on tänu nutiseadmete ulatuslikule levikule võimalik vähendada. Käesoleva lõputöö eesmärk oli projekteerida ja konstrueerida prototüüpseade, millega saaks uurida valguse mõõtmise võimalusi uut tüüpi tehnoloogilise lähenemisega. Töö tulemusena valmis akutoitel töötav prototüüpseade, millega on võimalik tajuda valguse intensiivsust mingis ruumipunktis ning millega saab kogutud andmeid juhtmevabalt edastada. Koostatud seadme edasiarendamine võimaldab jõuda uuendusliku tooteni.</td></tr>
<tr><td colspan="4">Märksõnad: Fotomeetria, valgusmõõtja, Bluetooth Low Energy</td></tr>
</table>

ABSTRACT

www.emu.ee
Eesti Maaülikool
Estonian University of Life Sciences

Estonian University of Life Sciences Kreutzwaldi 1, Tartu 51014		Summary of a Professional Higher Education Thesis	
Author: Jan Bogdanov		Speciality: Technotronics	

Title: Wireless accessory for a smartphone to measure illuminance and colour temperature			
Pages: 35	Figures: 17	Tables: 2	Appendixes: 3
Department: College of Technology Field of research: Project Supervisors: Tõnu Leemet, Tanel Ainla Place and date: Tartu, 2017			

Measuring the properties of light in our surrounding environment is important for various specialists. Currently, devices for measuring the properties of light exhibit certain limitations that can be reduced due to the with the help of smartphones and tablet computers. The aim of this thesis was to design and construct a prototype that can be used to study novel technological solutions for measuring certain properites of light. As a result, a battery-operated device, which can be used to sense the intensity of light at a point in space and which is capable of wireless communication, was constructed. Further development of the device will enable it to become an innovative product.

Keywords: Photometry, Light metering, *Bluetooth Low Energy*

Sisukord

Tähised ja lühendid

ISM - *Industrial, Scientific, Medical*, rahvusvaheliselt kokkulepitud raadiosagedusvahemik

BLE - *Bluetooth Low Energy*, raadiopersonaalvõrgu tehnoloogia

TDD - *Time Division Duplex*, dupleksrežiimi tüüp

FEC - *Forward Error Correction*, ennetav veaparandus

I^2C - *Inter-Integrated Circuit*, kahesuunaline kahesooneline järjestiksiin

USB - *Universal Serial Bus*, universaalne järjestiksiin

NRZ - *Non-Return-To-Zero*, kodeeringutüüp

$UUID$ - *Universally Unique Identifier*, globaalne identifikaator

LSB - *Least Significant Bit*, vähima kaaluga bitt

λ - elektromagnetkiirguse lainepikkus, nm

Φ_R - kiirgusvoog, W

Φ_v - valgusvoog, lm

E - kiiritustihedus, W/m^2

E_v - valgustatus, lx

A - pindala, m^2

c - valguse kiirus vaakumis, 299 792 458 m/s

h - Plancki konstant, 6,62607004x10^{-34} $J \cdot s$

k - Boltzmanni konstant, 1.38064852x10^{-23} J/K

T - absoluutne temperatuur, K

Sissejuhatus

Valgusallikatel on inimkonna jaoks kriitiline tähtsus – need võimaldavad inimestel toimida vähese või olematu päevavalguse tingimustes, kujundada endale sobivalt oma elu- ja töökeskkondi ning kasutada valgusallikaid oma erialaste eesmärkide saavutamiseks. Valguse olulisus toob esiplaanile vajaduse mõista ja mõõta inimeste igapäevaselt ning erialaselt kasutatavate valgusallikate ja valgustustingimuste omadusi. Uuritavad valguse omadused võivad olla erinevad – töökeskkonnaspetsialist võib olla huvitatud peamiselt tööruumi valgustatusest, kuid fotograaf võib oma töös vajada lisaks informatsiooni valguse värvustemperatuuri kohta [1, 2]. Tehnoloogia areng ja selle kättesaadavuse paranemine soodustab valguse omaduste mõõtmiseks kasutatavate tehniliste lahenduste uuendamise võimaluste uurimist.

Valguse värvustemperatuuri ning valgustatust mõõtvaid seadmeid on konstrueeritud aastakümneid, kuid käesoleva lõputöö raames läbiviidud taustauuringu käigus leiti peamiselt seadmeid, mille andur ning mõõdetud andmeid kuvav indikaator asusid füüsiliselt sama seadme küljes. Selliste mõõteseadmete arhitektuur ei võimalda sidestada mitme erineva anduri mõõtetulemusi ühe indikaatoriga, mistõttu ei ole võimalik mõõta ning vaadelda samaaegselt mitmes erinevas punktis valguse omadusi. Niisamuti on valdavalt tegemist eraldiseisvate seadmetega, mistõttu on vaja nende transportimiseks lisaruumi. Eranditeks on valguse parameetreid mõõtvad nutitelefonide lisaseadmed *Lumu Power* [3] ning *Cine Meter II* [4], mis on sõltuvad nutiseadme tootjast ega võimalda teostada mõõtmisi distantsilt. Nutiseadmele elektrooniliste lisade projekteerimist võib pidada perspektiivikaks seetõttu, et nutiseadmete kasutamine on levinud ning nende kasutajate arv on suurenemas [5, 6]. Lisaks on enamikel nutiseadmetel suhteliselt suur arvutusjõudlus, mistõttu on võimalik vähendada projekteeritava lisaseadme riistvaralisi nõudeid. Tänapäeval on kõigil nutiseadmetel riistvara, mis toetab raadiopersonaalvõrkude kasutamist ja seetõttu on näiteks *Bluetooth*-tehnoloogia abil võimalik elektroonilisi lisaseadmeid sidestada paljude nutiseadmetega. Nutiseadmete ekraanid sobivad hästi andmete kuvamiseks ja lisaseadme juhtimiseks – neil on suhteliselt kõrge eraldusvõime ning dünaamiline ulatus, millega on võimalik kuvada näiteks võrdlemisi detailseid graafikuid. Nutiseadmete ühenduvus internetiga võimaldab lisaseadmetega sidestatavat mobiilitarkvara lihtsamalt uuendada.

Lõputöö eesmärk

Kogu käesolevas lõputöös esitletava on koostanud töö autor. Lõputöö põhifookus on järgmistel prototüüpseadme projekteerimise ning konstrueerimise toimingutel.

- Prototüüpseadmele on projekteeritud järgmised riistvaralised alamsüsteemid: toitesüsteem, valgusandur koos signaalisobitusahela ning sobitatud signaali diskreetimisahelaga, andmehõiveseade, raadiopersonaalvõrguseade, valgusdioodindikaatorid ning trükkplaadil asetsev temperatuuriandur.
- Prototüüpseadme riistvaralised alamsüsteemid on trükkplaadil täielikult koostatud.
- Prototüüpseadme kõik riistvaralised alamsüsteemid peale temperatuurianduri on katsetatud.
- Prototüüpseadmele on projekteeritud järgmised mehaanilised osised: koostatud trükkplaati ning energiasalvestit ümbritsev põhikorpus, mille disain lähtub elektroonikast tulenevatest piirangutest, ning valgusjuhid pindmontaažvalgusdioodide valguse kiirgumiseks korpuse pinnalt.
- Prototüüpseadmele on 3D-prinditud esmasel korpusedisainil baseeruvad osised.
- Prototüüpseadme andmehõiveseadmele on kirjutatud püsivara, milles realiseeritakse andmesiinide ning valgusdioodide juhtimisalgoritmid, toitesüsteemi seadistamine ning valitud raadiopersonaalvõrgu põhifunktsioonide toimimine.

Loetletud põhieesmärkide saavutamiseks oli vajalik täita järgmised kaasnevad eesmärgid:

- valitud raadiopersonaalvõrgu käitamiseks vajalike teadmise omandamine,
- valguse omaduste mõõtmiseks vajalike taustateadmiste omandamine.

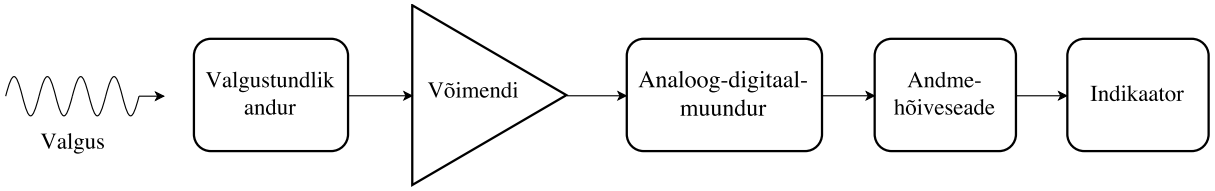
Lõputöö struktuur

Esimene peatükk annab ülevaate käesoleva lõputöö sisulise osa mõistmiseks olulistest eelteadmistest ning käsitleb fotoonika-alaseid alusteadmisi ja vajalikku läbilõiget lõputöös rakendatava elektroonika põhitõdedest. Teises peatükis keskendutakse lõputöö elektroonilise disaini kirjeldamisele. Kolmandas peatükis vaadeldakse prototüübi mehaaniliste

elementide konstrueerimise põhimõtteid. Neljandas peatükis käsitletakse süvitsi prototüübi tarkvaralisi lahendusi. Viiendas peatükis arutletakse teostatud projekti edasiste arendusvõimaluste üle. Diplomitöö lõpus toodud kokkuvõttes käsitletakse töö lõpptulemust.

1 Valdkonna ülevaade

Valgusena mõistetakse seda elektromagnetspektri lainepikkuste vahemikku, mida inimese silm tajub (390-700 nm). Erineva lainepikkusega elektromagnetkiirgust näeme erinevate värvustena ning inimsilma suhteline spektraalne tundlikkus on lainepikkusest sõltuv. [7] Objektile langeva valguse parameetreid, nagu näiteks valgustatust ning värvustemperatuuri, mõõtva seadme konstrueerimiseks on oluline tunda mõõdetavate füüsikaliste suuruste olemust ning mõõtmisi teostava seadme peamiste elektrooniliste ja optiliste alamkomponentide tööpõhimõtet. Elektroonilise valgusmõõteseadme põhimõtteskeem on toodud joonisel 1.1. Käesolevas peatükis antakse lühiülevaade olulisematest põhiteadmistest, mida töö autor on seadme konstrueerimisel kasutanud. Järgnevates alampeatükkides ei ole põhirõhk teemade süvitsi käsitlemisel, mistõttu on peatükkides kasutatud lihtsustusi ning keskendutud laiema taustinformatsiooni edasiandmisele.



Joonis 1.1: Valgus, mis langeb valgustundlikule andurile põhjustab selle välisahelas elektrivoolu. Harilikult on tekkinud elektriline signaal suhteliselt nõrk ning seetõttu on otstarbekas signaali võimendada. Andurilt saadava informatsiooni töötlemiseks mikroprotsessoriga on esmalt vajalik võimendatud elektriline signaal muuta analoog-digitaalmuunduris diskreetseks arvuliseks väärtuseks. Kui töödeldud informatsiooni on vaja kuvada, siis lisatakse süsteemi andmeid sobivalt esitav indikaator.

1.1 Kiiritustihedus ja valgustatus

Valgusallikate valgustusomaduste ning pindade valgustatuse mõõtmist uurivad radiomeetria ning fotomeetria teadusharud. Radiomeetriliste meetoditega uuritakse elektromagnetkiirgust terve elektromagnetlainete spektri ulatuses ning kirjeldatakse kiirgusvõimsuse jaotust ruumis. Fotomeetria uurib elektromagnetspektri nähtava valguse spektriosa ning

mõõdetud valgustustingimuste kirjeldamisel lähtutakse inimsilma spektraalsest tundlikkusest. Inimsilma spektraalne tundlikkus sõltub valgustustingimustest – valguse intensiivsusest ja lainepikkusest. Kokkuleppeliselt lähtutakse kahest inimsilma spektraalse tundlikkuse mudelist – fotoopilise nägemise ehk päevanägemise mudelist ning skotoopilise nägemise mudelist, mida on kohandatud inimsilma spektraalse tundlikkuse muutustele madalate valgustasemete tingimustes. [8]
Radiomeetria põhisuuruseks nimetatakse kiirgusvoogu Φ_R, mis on võrdeline elektromagnetkiirguse kantava energiahulgaga ajaühikus ning mille mõõtühik on W. Kiiritustiheduseks E nimetatakse pinnale A langevat kiirgusvoogu. Kiiritustiheduse mõõtühik on $\frac{W}{m^2}$ ning E avaldub kui [7]

$$E = \frac{kiirgusvoog}{pindala} = \frac{d\Phi_R}{dA}. \tag{1.1}$$

Fotomeetria põhisuuruseks nimetatakse valgusvoogu Φ_v, mis on võrdne kiirgusvooga Φ_R, kuid mille väärtus on kohandatud inimsilma spektraalse tundlikkuse mudeliga. Valgusvoo Φ_v mõõtühik on luumen. Valgustatuseks E_v nimetatakse pinnale A langevat valgusvoogu. Valgustatuse mõõtühik on luks (lx) ning E_v avaldub kui [7]

$$E_v = \frac{valgusvoog}{pindala} = \frac{d\Phi_v}{dA}. \tag{1.2}$$

1.2 Värvustemperatuur

Valgustatud objekti tajutav värvitoon sõltub selle objekti värvusest, objekti spektraalsest peegelduvusest ning antud objekti valgustava valgusallika spektraalsest võimsusjaotusest. Värvustemperatuuri kirjeldatakse musta keha kiirguse abil. Must keha on idealiseeritud objekt, mis neelab kogu temale langeva valguse (elektromagnetkiirguse) olenemata valguse lainepikkusest või valguse langemisnurgast. Energia jäävuse seaduseset lähtuvalt pab must keha ka kiirgama. Musta keha kirkus ja kiirguse lainepikkuste jaotus ehk spektraalne kirkus on määratud ainult musta keha temperatuuriga. Musta keha kiirguse spektraalset kirkust kirjeldab Plancki kiirgusseadus [7]

$$L(\lambda) = \frac{2hc^2}{\lambda^5} \cdot \frac{1}{e^{\frac{ch}{\lambda kT}} - 1}, \tag{1.3}$$

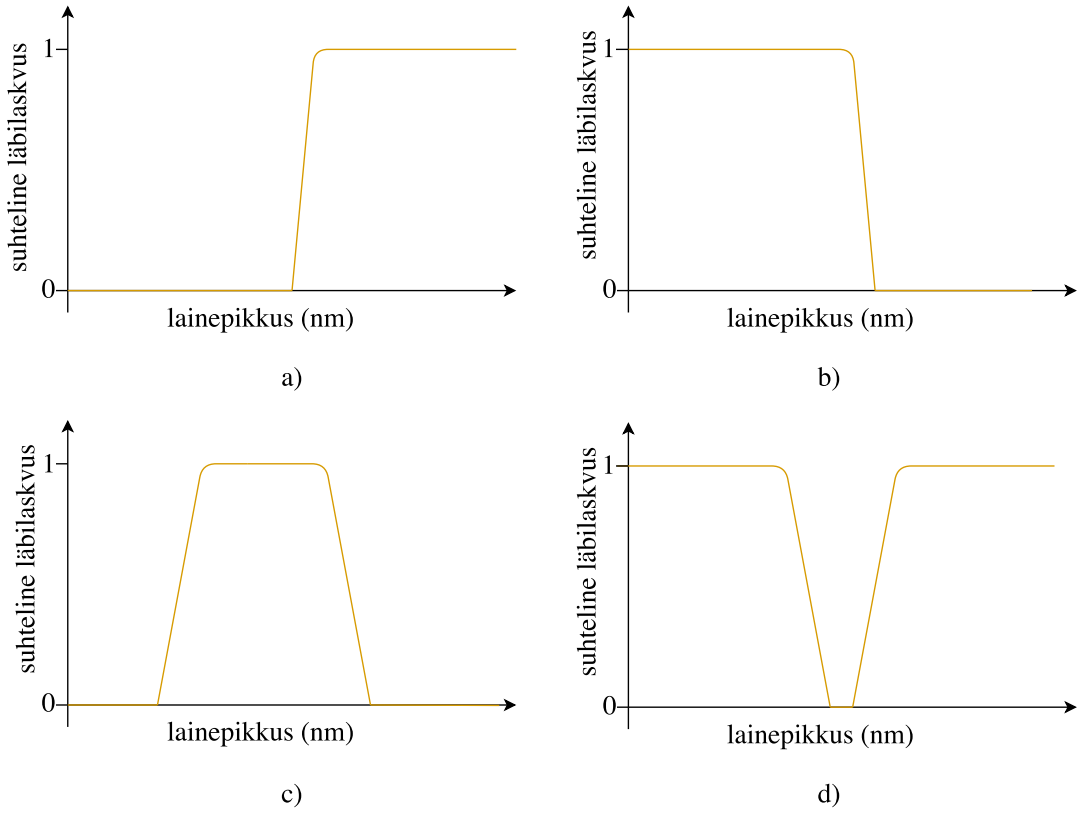
kus h on Plancki konstant, k on Boltzmanni konstant, c on valguse kiirus vaakumis, λ on lainepikkus ning T on absoluutne temperatuur. Plancki kiirgusseadusest tuleneb Wieni nihkeseadus, mis kirjeldab millisel lainepikkusel on musta keha kiirguse maksimum ning avaldub kui

$$\lambda_m T = \frac{ch}{4.965k} = 0.0028978\, m{\cdot}K. \tag{1.4}$$

Temperatuuriga T musta kehaga sama värvi valgusallika värvustemperatuur on T juhul, kui valgusallikas on mustkiirgur. Kui valgusallikas ei ole mustkiirgur, kuid tema kiirgus on lähedane musta keha kiirgusele, siis nimetatakse valgusallika temperatuuri T korreleeritud värvustemperatuuriks *(correlated colour temperature)*. Käesoleva töö järgnevates osades mainitud värvustemperatuuri all mõeldakse korreleeritud värvustemperatuuri, välja arvatud juhul, kui on mainitud teisiti. [7]

1.3 Optilised filtrid

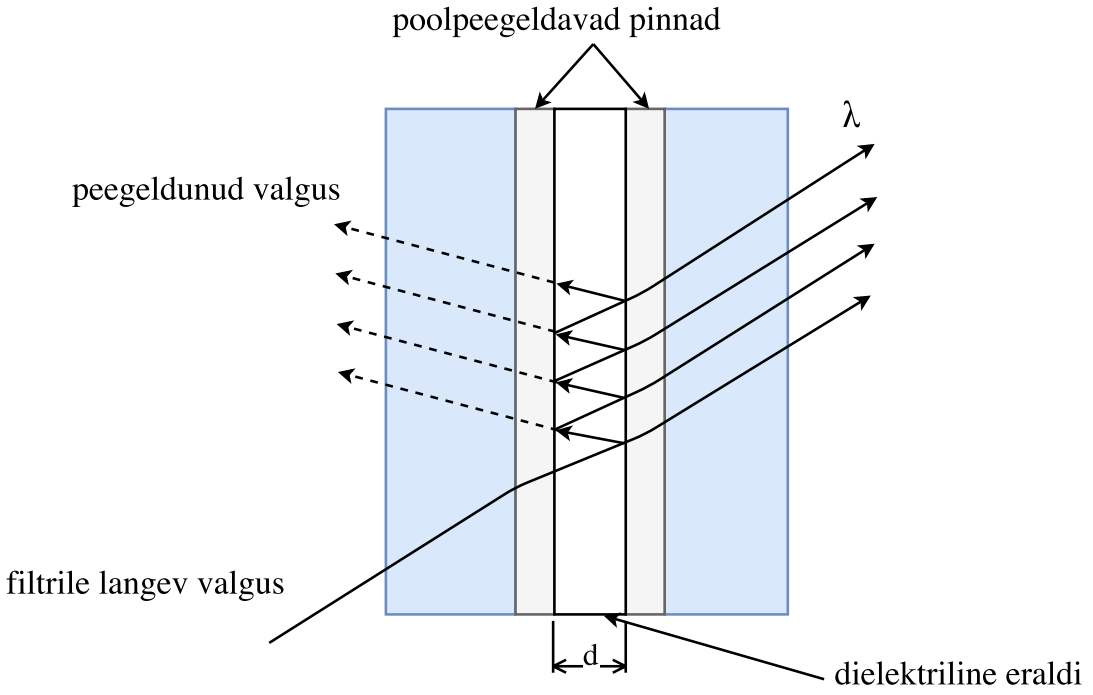
Optiline filter on seade, mille valguse läbilaskvus on sõltuv filtrile langeva valguse lainepikkusest, polarisatsioonist, ruumijaotusest või nõrgendab filter valgust ühtlaselt terve määratud lainepikkusvahemiku ulatuses. Optilisi filtreid jaotatakse läbilaskvuskõvera kuju järgi ribafiltriteks, mis lasevad läbi vaid kindlas lainepikkusvahemikus valgust, sälkfiltriteks, mis ei lase läbi kindlas lainepikkusvahemikus valgust ning servfiltriteks, mis ei lase läbi valgust, mis on suurem või väiksem mingist lainepikkusest alates. Läbilaskvuskõveraid kujutavad graafikud on näidatud joonisel 1.2. Optilisi filtreid kasutatakse, kui optilisest süsteemist on vaja eemaldada või nõrgendada mingis lainepikkusvahemikus olev valgus, nagu näiteks päikeseprillides, fluorestsentsmikroskoopides, spektrianalüsaatorites või laserite käitamisel. [9]



Joonis 1.2: Diagrammil a on kujutatud kõrgpäässervfiltri, diagrammil b on kujutatud madalpäässervfiltri, diagrammil c on kujutatud ribafiltri ning diagrammil d sälkfiltri läbilaskvuskõveraid.

Interferentsfilter

Interferentsfiltriteks nimetatakse optilisi filtreid, mille toime põhineb interferentsinähtusel. Soovitud lainepikkuse läbilaskmiseks asetseb õhuke dielektriline materjal kahe poolpeegeldava pinna vahel. Interferentsifiltrite filtreerimisomadused muutuvad kui pealelangev valgus ei taba filtrit pinnaga risti. [10]
Interferentsfiltrid saavutavad neelavate filtritega võrreldes suhteliselt suure tõusuga läbilaskvuskõveraid [11]. Joonisel 1.3 on näidatud interferentsfiltri põhimõtteskeem.



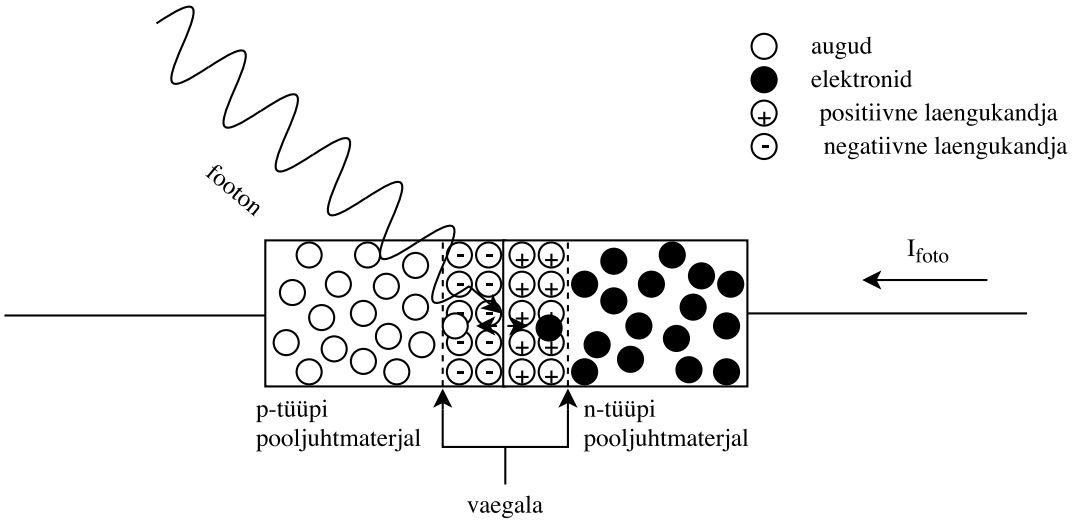
Joonis 1.3: Dielektrilise interferentsfiltri poolpeegeldavad pinnad koosnevad vaheldumisi suure ning väikese murdumisnäitajaga dielektrilistest kihtidest, mille paksus on $\frac{\lambda}{4}$, kus λ on soovitud lainepikkus. Dielektrilise eraldi paksus on $\frac{\lambda}{2}$. Filtrile langev valgus peegeldub ning interfereerub poolpeegeldavate pindade vahel. [9]

1.4 Fotodetektorid

Fotodetektor on optiline andur, mille peamine omadus on võime reageerida temale pealelangevale valgusele. Optiliste signaalide detekteerimisel eristatakse termilistel mõjudel, valguslainete ja mittelineaarsete materjalide vastastikmõjudel ning fotoelektrilistel mõjudel põhinevaid nähtusi. Termilistel mõjudel põhinevate fotodetekorite väljundparameeter, näiteks fotodetektori takistus või mahtuvus, on sõltuv temperatuurimuutusest, mida põhjustab detektorile langev elektromagnetkiirgus. Termilistel mõjudel põhinevad seadmed on näiteks püroelektriline radiomeeter ja bolomeeter. Valguslainete ja mittelineaarsete materjalide vastastikmõjudel põhinevates fotodetektorites sõltub anduri optiline väljundsignaal tema optilisest sisendsignaalist. Seda nähtust rakendatakse näiteks optilistes parameetrilistes võimendites. Fotoelektrilisel efektil baseeruvale fotodetektorile langevad footonid tekitavad atomaarse tasandi vastastikmõjude tõttu laengukandjaid. Sellepärast on selliste fotodetektorite väljundiks elektrivool. [12]

Pn-tüüpi fotodetektor

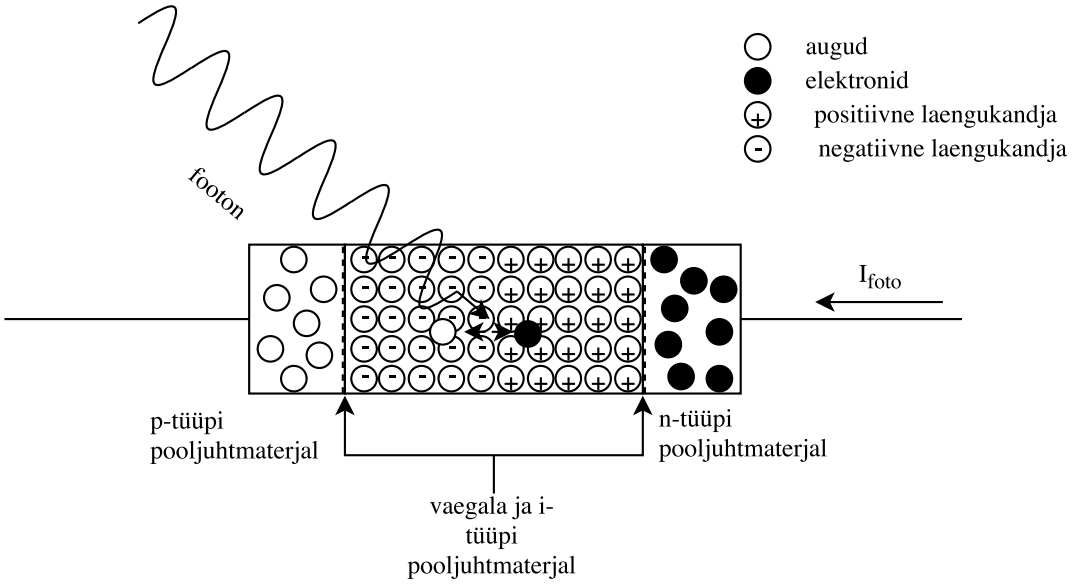
Pn-tüüpi fotodetektor ehk pn-fotodiood on fotoelektrilisel efektil põhinev pooljuhtseadis, mis konverteerib seadme pooljuhtmaterjalile langeva valguse elektrivooluks. Pn-fotodiood koosneb ühest või mitmest pooljuhtmaterjalist, mis on jaotatud lisandainete aatomite pooljuhtmaterjali legeerimise teel kaheks piirkonnaks: negatiivsete laengukandjatega dopeeritud n-tüüpi pooljuhiks ning elektronaukudega – elektroni puudumisest tingitud positiivsete laengukandjatega – dopeeritud p-tüüpi pooljuhiks. N- ja p-tüüpi pooljuhtmaterjalide üleminekuala nimetatakse pn-siirdeks. Pn-siirde moodustumisel difundeeruvad n-tüüpi pooljuhtmaterjalis olevad elektronid ning p-tüüpi pooljuhtmaterjalis olevad augud. Pn-siiret läbivad augud ja elektronid rekombineeruvad, kuniks pn-siirde pooltele kogunevate erinimeliste laengute kontsentratsioon on piisav takistamaks laengukandjatel pn-siirde läbimist. Rekombineerumise tulemusena tekkinud ruumlaengud moodustavad n-tüüpi pooljuhist p-tüüpi pooljuhi suunalise elektrivälja.
Pn-fotodioodi vaegalale langev footon põhjustab elektron-auk paari moodustumise, mis liigutatakse elektrivälja toimel vaegala äärepiirkondadesse. Vaegalast väljunud laengukandjad liiguvad pn-tüüpi fotodioodi väljaviikudesse ning tekib elektrivool (joonis 1.4). [12]



Joonis 1.4: Pn-tüüpi fotodioodi vaegalale langev footon tekitab elektron-auk paari, mis vaegala vahel oleva elektrivälja mõjul teineteisest eralduvad ning seejärel vaegala pooluste poole liiguvad. Tekkinud laengukandjate liikumine põhjustab pn-tüüpi fotodioodiga ühendatud välisahelas elektivoolu I_{foto}.

Pin-tüüpi fotodetektor

Pin-tüüpi fotodetektoriks nimetatakse pooljuhtdioodi, mille p- ja n-tüüpi pooljuhtidest osade vahel on legeerimata või nõrgalt legeeritud i-tüüpi pooljuhtmaterjal [13]. Pin-fotodioodi vaegala katab peaaegu terve i-tüüpi pooljuhtmaterjali ulatuse. Kuna i-tüüpi pooljuhtmaterjal sisaldab suhteliselt vähe laengukandjaid on pin-tüüpi fotodioodi vastupingestamisel võimalik tekitada suurema ulatusega vaegala kui pn-tüüpi fotodioodil (joonis 1.5). Suurema vaegala tõttu on pin-tüüpi fotodioodile langeval footonil suurem tõenäosus tekitada elektron-auk paar, mistõttu on pin-fotodiood valguse muundamisel elektrivooluks tõhusam kui pn-fotodiood. Lisaks on pin-fotodioodil erinimeliste laengute vahelise ala suurema kauguse tõttu väiksem siirdemahtuvus, kui pn-fotodioodil. [12]



Joonis 1.5: Pin-tüüpi fotodiood on pn-fotodioodist laiema vaegala tõttu suurema kvantsaagisega ning pealelangeva valguse muundamisel elektron-auk paariks efektiivsem, kuna laengukandjad liiguvad ligikaudu terve vaegala ulatuses oma maksimaalse kiirusega.

1.5 Raadiopersonaalvõrgu tehnoloogia *Bluetooth Low Energy*

Bluetooth Low Energy, edaspidi BLE, on ettevõtte *Bluetooth Special Interest Group* arendatud raadiopersonaalvõrgu tehnoloogia, mis on disainitud elektroonikaseadmete vahelise madala energiatarbega raadiosideühenduste loomiseks. [14]

Bluetooth Low Energy füüsiline kiht

BLE seadmed toimivad litsenseerimata 2,4 GHz ISM sagedusribas sagedusvahemikus 2400 - 2483,5 MHz, mis on jaotatud 40 raadiokanaliks kesksagedustega 2402 + k · 2 MHz, kus k = 0, ..., 39. BLE 4.1 minimaalne edastusvõimsus on 0.01 mW ning maksimaalne edastusvõimsus on 10 mW. Interferentsi mõjude vähendamiseks, kasutavad BLE transiiverid sagedushüplemist. BLE standard spetsifitseerib kaks modulatsioonimeetodit [15]:

- LE 1M – kohustuslik binaarset sagedusmodulatsiooni kasutav modulatsiooniskeem BLE transiiveritele, mis edastab kodeerimata andmevoogu kiirusega 1 Mb/s. Lisaks on LE 1M võimalik valikuliselt asendada LE Coded skeemiga, mis edastab kodeeritud andmevoogu, kus osa paketi päisest on kodeeritud kiirusega 125 kb/s ning paketi andmed on kodeeritud kiirusega 125 kb/s või 500 kb/s.
- LE 2M – kohustuslik binaarset sagedusmodulatsiooni kasutav modulatsiooniskeem BLE transiiveritele, mis edastab kodeerimata andmevoogu kiirusega 2 Mb/s.

LE 1M ja LE 2M rakendavad *Time Division Duplex*-režiimi [15], ehk TDD-d, mis eraldab samas sagedusribas toimiva andmete üleslingi andmete allalingist jagades neile eraldi edastamiseks ajaaknad. TDD meetod võimaldab emuleerida poolduppleksrežiimis toimiva andmesidega täisduppleksrežiimis andmeedastust ning on otstarbekas olukordades, kus üleslingi ja allalingi andmeedastuskiirused on teineteisest erinevad [16].

Bluetooth Low Energy raadolingi kiht

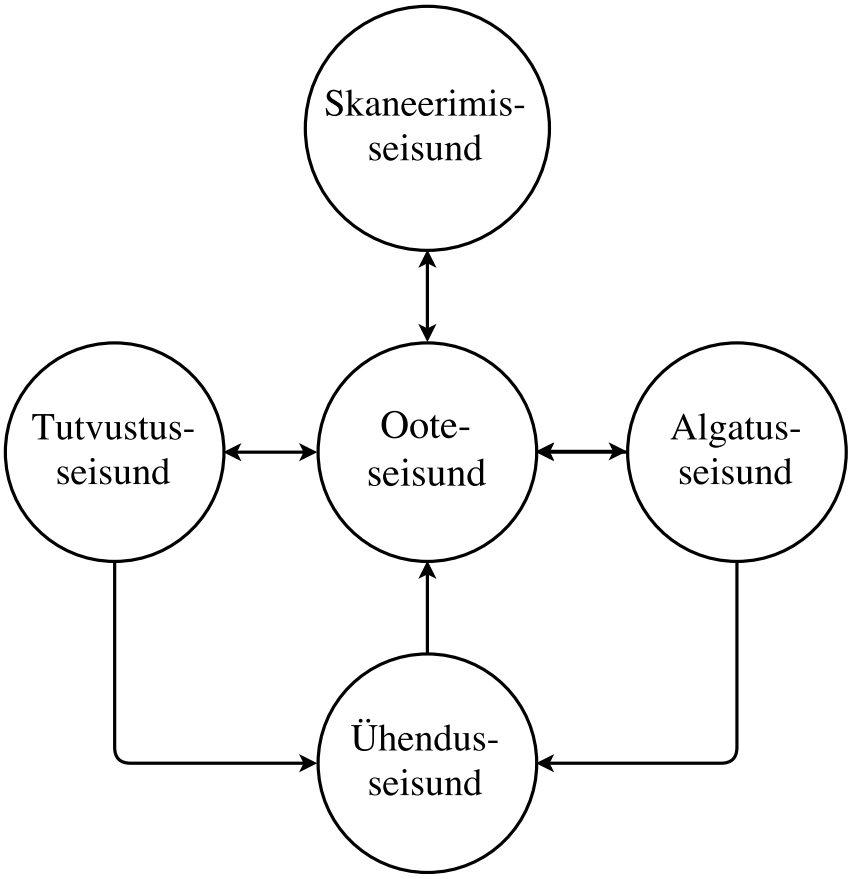
BLE raadiolingi kihti võib käsitleda olekumasinana. Olekumasinal on korraga ainult üks aktiivne seisund. Ühel BLE raadiolingi kihil võib olla mitu olekumasinat. millel on viis olekut (joonis 1.6) [15].

- *Standby State* ehk ooteseisund – ooteseisundis raadiolingi kiht ei edasta ega võta vastu andmepakette. Ooteseisundisse võib siseneda kõikidest teistest olekutest.
- *Advertising State* ehk tutvustusseisund – tutvustusseisundis edastatakse tutvustuskanalis andmepakette ning tõenäoliselt kuulatakse ja vastatakse tutvustuskanalis edastatud andmepakettide vastustele. Tutvustusseisundisse on võimalik siseneda ooteseisundist.

- *Scanning State* ehk skaneerimisseisund – skaneerimisseisundis kuulatakse tutvustuskanalis edastatavaid andmepakette. Skaneerimisseisundisse sisenetakse ooteseisundist.

- *Initiating State* ehk algatusseisund – algatusseisundis kuulatakse andmepakette, mida edastavad tutvustuskanalis spetsiifilised seadmed eesmärgiga neile ühendumisseisundisse sisenemiseks vastata. Algatusseisundisse on võimalik siseneda ootseisundist.

- *Connection State* ehk ühendusseisund – ühendusseisundisse on võimalik siseneda tutvustus- või algatusseisundist. Ühendusseisundis eristatakse ülema ning alluva rolle. Algatusseisundist ühendusseisundisse astuja on ülema rollis ning tutvustusseisundist ühendusseisundisse astuja on alama rollis. Ülema rolli täitva raadiolingi kiht määrab andmevahetuse ajastused ning suhtleb alama rolli täitva raadiolingi kihiga. Alama rollis seade suhtleb ühe ülema rollis seadmega.

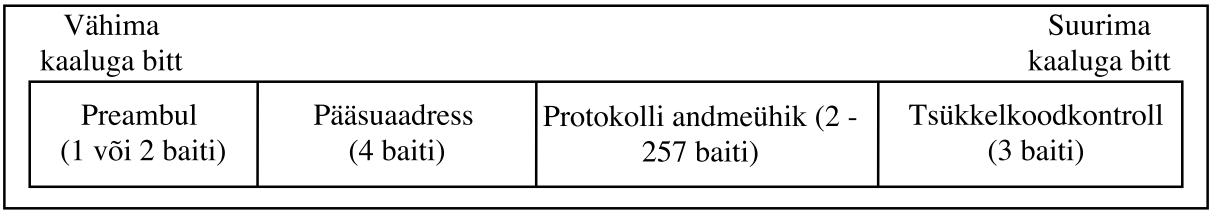


Joonis 1.6: *Bluetooth Low Energy* raadiolingi kihi olekudiagramm [15].

BLE füüsilise kihi 40 raadiokanalit on jaotatud kolmeks füüsiliseks kanaliks – tutvustuskanaliks, perioodiliseks kanaliks ning andmekanaliks, millest igaüks on unikaalselt indekseeritud. Tutvustuskanal kasutab seadmete skaneerimiseks, ühenduste loomiseks ning andmete edastamiseks kõiki 40 raadiokanalit, millest kolm on kategoriseeritud põhitutvustuskanaliteks ning 37 kõrvaltutvustuskanaliteks. Andmekanal ning perioodiline kanal kasutavad andmete edastamiseks neidsamu 37 raadiokanalit, mida kasutavad kõrvaltutvustuskanalid. [15]

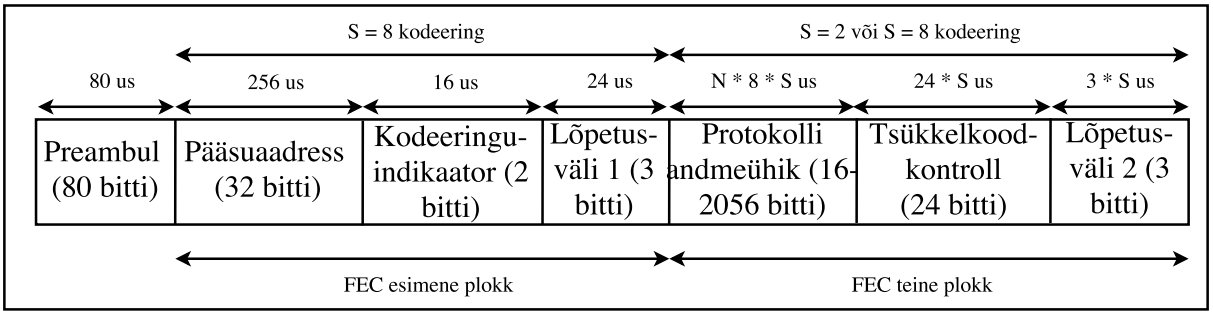
Kaks omavahel andmevahetust pidavat seadet peavad jagama sama füüsilist kanalit ja seega peavad nende transiiverid olema seadistatud samale raadiokanalile. Raadiokanalite piiratud hulga tõttu mitme erineva BLE seadme samaaegsest suhtlemisest tingitud võimalike häirete vähendamiseks lisatakse igal füüsilisel kanalil edastatava andmepaketi algusesse pääsuaadress, mida kasutatakse kõigi samal füüsilisel kanalil asuvate seadmete orienteerimiseks. Raadiolingi kiht kasutab igal ajahetkel korraga üht füüsilist kanalit. [15]

Raadiolingi kihi esimene paketiformaat on spetsifitseeritud mittekodeeritud füüsilistele kihtidele LE 1M ja LE 2M ning seda kasutavad tutvustus- ja andmekanalis edastatavad paketid. Andmevahetuse käigus edastatakse vähima kaaluga bitt esimesena ning terve pakett edastatakse kas 1 Mbit/s või 2 Mbit/s sagedusmodulatsiooniga. LE 1M ja LE 2M füüsiliste kihtide pakettide edastamiseks kulub 44 µs - 2120 µs. Joonisel 1.7 on näidatud raadiolingi kihi paketiformaat mittekodeeritud füüsilistele kihtidele LE 1M ja LE 2M. [15]



Joonis 1.7: LE 1M füüsilise kihi preambuli pikkus on 1 bait ja LE 2M füüsilise kihi preambuli pikkus on 2 baiti. Protokolli andmeühiku pikkus ja formaat sõltub sellest, kas pakett edastatakse tutvustuskanalil või andmekanalil. [15]

Raadiolingi kihi teine paketiformaat on spetsifitseeritud kodeeritud füüsilisele kihile LE Coded ning seda kasutavad tutvustus- ja andmekanalis edastatavad paketid. LE Coded paketiformaat kasutab 1 Mbit/s sagedusmodulatsiooni ning LE Coded pakettide edastamiseks kulub 462 µs - 17040 µs. Joonisel 1.8 on näidatud raadiolingi kihi paketiformaat kodeeritud füüsilisele kihile LE Coded. [15]



Joonis 1.8: LE Coded füüsilise kihi preambul ei ole kodeeritud. Kodeeringuindikaator FEC määrab teise ploki paketipikkuse ning edastamisaja. [15]

2 Seadme mehaaniline disain

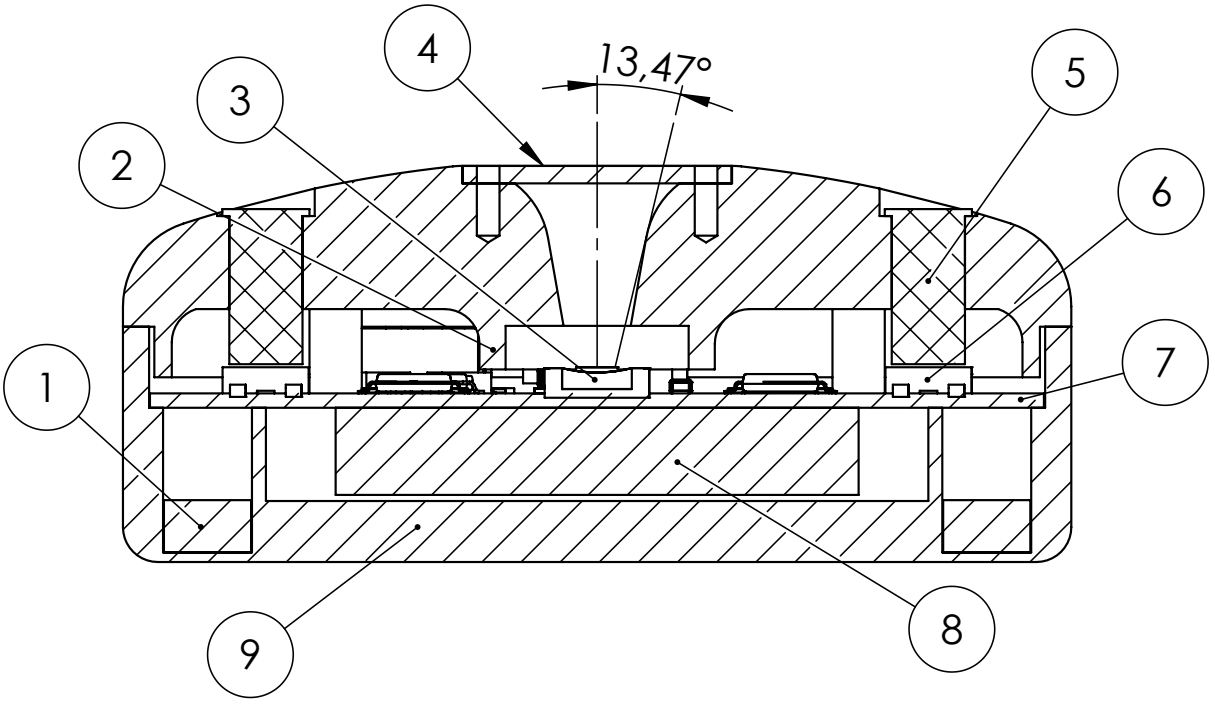
Käesolevas peatükis kirjeldatakse projekteeritud ning koostatud mehaanilise prototüübi konstruktsiooni. Trükkplaadi ning liitiumioon-aku ümbritsemiseks, seadme vertikaaltasapindadele kinnitamisvõimaluste uurimiseks ning andurile sobivate valgustustingimuste loomiseks projekteeriti prototüübile korpus. Seadme mehaanilise osa disainimisel kehtis eeldus, et korpuse esimene prototüüp konstrueeritakse arendusprotsessi kiirendamiseks 3D-printimise meetodil ning projekteerimise hilisemates arenguetappides ei välistata korpuse osiste tootmist metallist. Disainitud korpus pidi täitma järgmisi tingimusi:

- korpus peab piirama andurile langeva valguse nurka, et tagada tajuril asetsevate interferentsfiltrite spetsifitseeritud summutusomadused,
- korpus peab valgustorude abil võimaldama trükkplaadil asetsevate valgusdioodindikaatorite kiirguse kandmise korpuse pinnalt emiteeruvaks,
- korpus ja selle elemendid peavad kaitsma trükkplaadil asetsevaid komponente mehaaniliste välismõjurite eest,
- korpus peab välistama trükkplaadil asetsevate valgusdioodindikaatorite emiteeritava kiirguse mõju tajurile.

Korpus koosneb kahest põhielemendist – esikaanest ning tagakaanest, mis kinnitatakse teineteise külge M3 sisekuuskantpoltide ning mutritega. Esi- ja tagakaanes on riskülikukujulise profiiliga kaks ava, millest üks on mikro-tüüpi USB pesale ligipääsemiseks ning teine on seadme nuppudele ligipääsemiseks. Nupudetail kinnitatakse esi- ja tagakaanel asuva kahe kontsentrilise mitteläbiva ava vahele paigutatud tüübliga. Korpuse koostu läbilõige, mis illustreerib seadme põhielemente, on näidatud joonisel 2.1. Kõik nimetatud elementide tehnilised joonised on näidatud lisas A.

Korpuse esikaas on monoliitne element, mis jääb sellele trükkplaadi poolele, kus asub tajur. Esikaanel on neli keermestamata läbivat ava poltide jaoks ning kaks keermestamata läbivat ava valgusjuhtide monteerimiseks. Lisaks on esikaanele projekteeritud

kaitseklaasiga kaetav avaus valguse pääsemiseks tajurini ning süvend koos avadega kaitseklaasi kinnitamiseks. Esikaas hõlmab ka tajurit ümbritsevat korpuse osa, mis on mõeldud takistama valgusdioodide kiirguse levimist tajurini.
Korpuse tagakaas on terviklik detail, milles on ettenähtud ruum seadme liitiumioon-aku ning trükkplaadi hoidmiseks. Liitiumioon-aku jääb asetsema termistoripoolsele trükkplaadi osale. Lisaks on tagakaanel neli keermestamata läbivat ava esi- ja tagakaane kinnitamiseks poltidega ning neli keermestamata mitteläbivat ava viiemillimeetrise diameetriga neodüümmagnetite hoidmiseks. Joonisel 2.2 on näidatud korpuse mudeli kujutist.



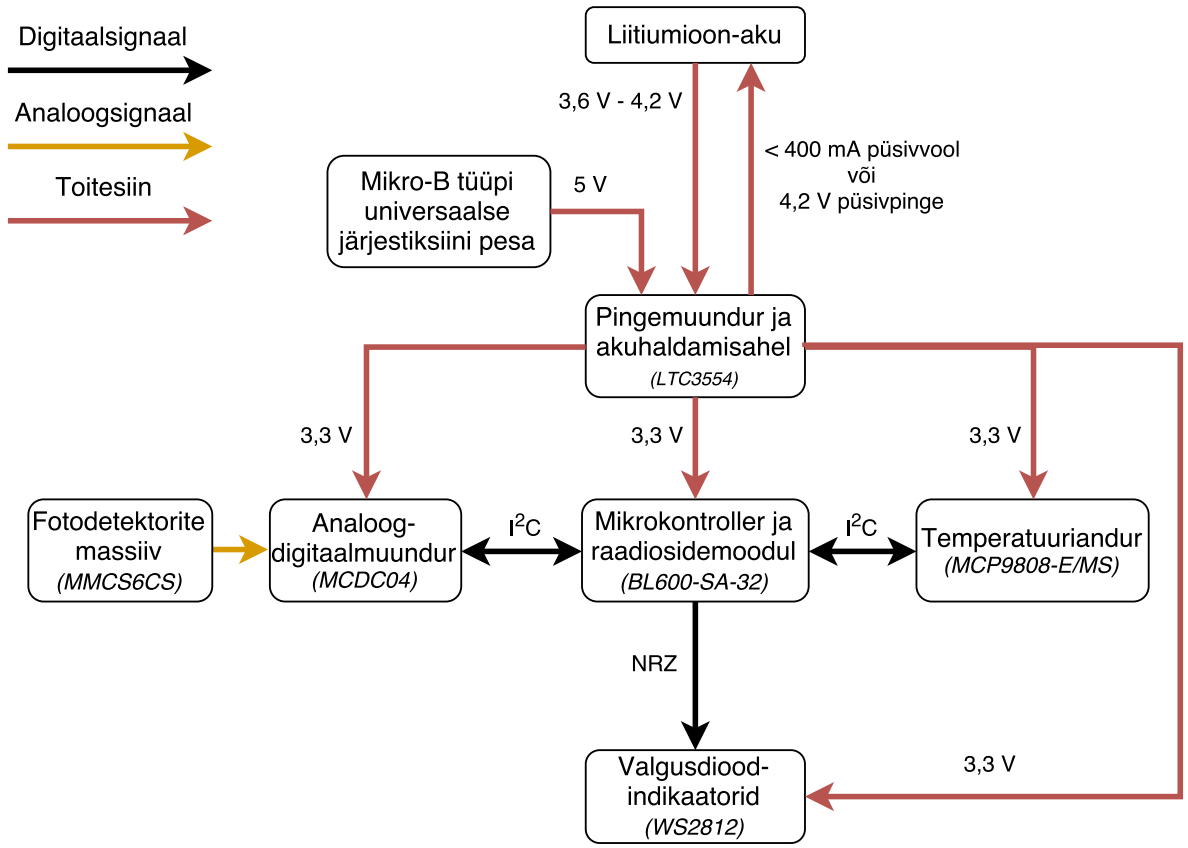
Joonis 2.1: Korpuse koostu läbilõike põhielemendid – 1) neodüümmagnet, 2) andurit valgusdioodindikaatoritest eraldav esikaane krae, 3) valgustundlik andur, 4) kaitseklaas, 5) valgustoru, 6) valgusdioodindikaator, 7) trükkplaat, 8) liitiumioon-aku, 9) korpuse tagakaas.

Joonis 2.2: Korpuse mudeli eestvaade – valgusdioodindikaatorite asetus võimaldab vaatlejale edastada informatsiooni seadme oleku või töörežiimi kohta. USB-pesa asetus võimaldab seadet laadida ning kasutada samaaegselt.

3 Seadme riistvaraline disain

Käesolevas peatükis kirjeldatakse projekteeritud ning koostatud riistvaralise prototüübi kasutatavaid elektroonikakomponente ja komponentide valiku põhimõtteid ning võimekusi. Seadme elektroonika topoloogia, mis on näidatud joonisel 3.1, aluseks on võetud joonisel 1.1 kujundatud põhimõtteskeem. Kõik seadme elektroonikaskeemid on näidatud lisas B. Seadme trükkplaadidisain on teostatud neljakihilisele trükkplaadile ning riistvaraliste komponentide selekteerimisel on lähtutud järgmistest kriteeriumitest:

- komponentide funktsionaalsuse sobivus projekteeritava seadme ülesannete täitmiseks,
- komponentide voolutarbe sobivus akutoitel põhineval seadmel kasutamiseks,
- komponentide kõrge integreerituse tase, et vähendada nende rakendamiseks vajalike diskreetsete komponentide hulka ning seega vähendada projekteeritava trükkplaadi pindala ja
- komponentide omavaheline sobivus, et tagada homogeensem riistvaraline topoloogia.



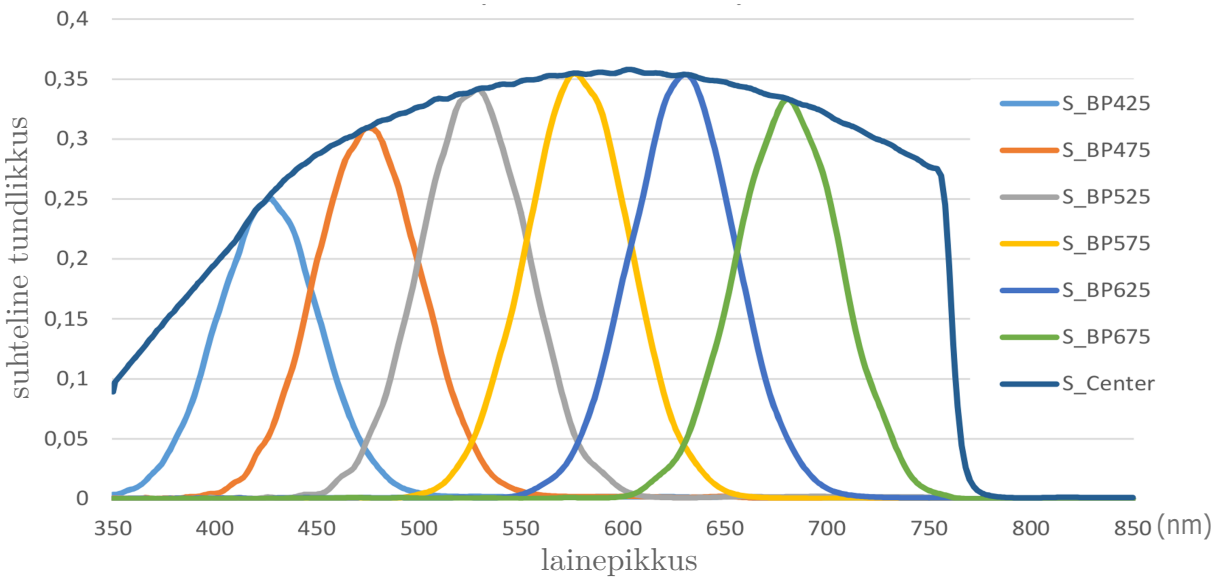
Joonis 3.1: Elektriahela topoloogia elektritoitesiin on spetsifitseeritud 3,3-voldisele pingele. Mikrokontrolleri suhtlus analoog-digitaalmuundurite ja temperatuurianduriga on realiseeritud I^2C protokollistikuga. Mikrokontroller edastab valgusdioodindikaatormoodulitele käske üle ühejuhtmeliidese kasutades mittenulltagastusega kodeeringut. Liitiumioon-akude laadimiseks vajalik elektrienergia saadakse standardse Mikro-B tüüpi universaalse järjestiksiini pesa kaudu.

3.1 Mitmikvärvitajur MMCS6CS

Seadmel kasutatake mitmikvärvitajurit MMCS6CS, mida peamiselt iseloomustab spektraalne tundlikkus üle terve nähtava valguse spektriosa. Komponent koosneb kokku kahekümnest pin-tüüpi fotodioodist ja korpusest. Fotodioodid asetsevad kaitseklaasi all komponendi pinnal 2 mm diameetriga pindalal ning on kaetud interferentsifiltritega, tekitades nii joonisel 3.2 näidatud suhtelise spektraalse tundlikkuse graafiku. Komponendi spektraalne tundlikkus võimaldab seda rakendada lisaks üldotstarbelistele värvuse mõõtmistele ka valguse spektri mõõtmistes, kasutades selleks matemaatilisi mudeleid. Lisaks on võimalik komponendiga tuvastada metameerilisi värve. [17]

Mitmikvärvitajuri MMCS6CS interferentsifiltrite omadused ei sõltu temperatuurist, kuid

komponendil asuvate fotodioodide tundlikkus ning tumevool sõltuvad, mistõttu on komponendi kasutamisel muutuva temperatuuriga keskkondades vajalik anduri mõõtetulemuste kasutamisel arvestada temperatuuri mõjudega. Interferentsifiltrite olemusest tulenevalt on anduri valgustundlikule piirkonnale pealelangeva valguse nurk maksimaalselt 10 kraadi. [17]



Joonis 3.2: Anduri kuue ribafiltritega pin-tüüpi valgusdioodi tundlikkused katavad lainepikkusvahemiku ligikaudu nähtava valguse ulatuses. Anduri laiema läbilaskvusega filtriga sobitatud valgusdiood hõlmab endas lisaks veel osa lähiinfrapunakiirgusest ning lähiultraviolettkiirgusest. [17]

3.2 Analoog-digitaalmuundur MCDC04

Anduri tekitatud fotovoolude võimendamiseks ja diskreetimiseks kasutatakse kaht analoog-digitaalmuundurit MCDC04. MCDC04 on 16-bitine neljakanaliline üle I^2C juhitav integraallülitus, mida rakendatakse vooluväljundiga optiliste andurite keskmise fotovoolu täppismuundamiseks digitaalsele kujule. Integraallülitus MCDC04 võimaldab I^2C protokolli kasutades muuta sisendvoolu integreerimisaega ja viiteoolu. Seega väljendub analoog-digitaalmuunduri MCDC04 ülekandefunktsioon kui [18]

$$TULEM = \frac{I_{sisend}}{I_{viide}} \cdot t_{INT} \cdot f_{TAKT}, \tag{3.1}$$

kus $TULEM$ on muundamise digitaalne väljund, I_{sisend} on muunduri kanali keskmine sisendvool muundamise aja jooksul, I_{viide} on viitevool, t_{INT} on muundamise ajaline kestus ning f_{TAKT} on muunduri taktsagedus. [18]

Sõltuvalt rakendusvajadusest on analoog-digitaalmuundurit MCDC04 võimalik käitada kolmes töörežiimis: [18]

- Käsurežiim – üle I^2C edastatud käsu peale teostab integraallülitus ühe mõõtmise ning muundamise.
- Pidevmõõterežiim – üle I^2C edastatud käsu peale teostab integraallülitus tsükliliselt mõõtmis- ning muundamisprotsesse, kuni üle I^2C edastatakse lõpetamiskäsklus.
- Sünkroonrežiimid – võimaldab sünkroniseeritud mõõtmis- ning muundamiprotsesse juhtida integraallülituse SYN väljaviigul toimuva langeva frondi abil.

3.3 Mikrokontroller ja raadiosidemoodul BL600-SA-32

Mikrokontroller ja raadiosidemoodul BL600-SA-32 on ettevõtte Laird projekteeritud seade, mis põhineb ettevõtte Nordic Semiconductor disainitud mikrokiibil nRF51822. BL600-SA-32 valiti eesmärgiga vältida integraalskeemiga nRF51822 kaasnevate raadiosidespetsiifiliste elementide arendamiseks kuluvat ajahulka. Kuna moodul BL600-SA-32 põhineb integraallülitusel nRF51822, siis käsitletakse käesolevas alampeatükis seadet nRF51822 ning lähtutakse komponendi nRF51822 andmelehest.

Integraallülitus nRF51822 koosneb 2,4 GHz transiiverist, millega teostatakse mitmeprotokollilist *Bluetooth Low Energy* raadiopersonaalvõrgutehnoloogiat, ning ARM Cortex-M0 32-bitisest protsessorist. Lisaks on Nordic integraallülitusele nRF51822 välja töötanud *Bluetooth Low Energy* protokollide realiseerimiseks tarkvarapaketi *SoftDevice S130*, mis pakub rakendusliidest *Bluetooth Low Energy*-süsteemide kiiremaks arendamiseks.

3.4 Pingemuundur ja akuhaldamisahel LTC3554

Integraallülitus LTC3554 on madala võimsusega elektritoite haldusseade mis on mõeldud üheelemendilistel liitiumioon- või liitiumpolümeerakudel põhinevate toitesüsteemide reguleerimiseks. Komponent tagab elektritoitesüsteemile järgmised võimekused:

- kaks reguleeritava väljundpingega kuni 200 mA väljundvooluga sünkroonset pinge impulss-stabilisaatorit,
- USB-l põhinev liitiumioon- või liitiumpolümeeraku laadimisahel, koos automatiseeritud koormusjaoturiga,
- reguleeritav laadimisvool vahemikus 0 - 500 mA ning laadimisvoolu piiramine juhul, kui tarbimisvool ning laadimisvool ületaks seadistatud laadimisvoolu ülempiiri,
- automaatne laadimisvoolu piiramine 10 % peale seatud väärtusest, kui laetava aku elektromotoorjõud on alla 2,9 voldi,
- integreeritud seadme käivitus- ja lähtestamisahel surunupplüliti abil,
- negatiivse temperatuurikoefitsendiga termistori kasutamise võimalus, et katkestada laadimine, kui akutemperatuur on liiga kõrge või liiga madal,
- automaatne laadimise taasalustamine, kui seade on ühendatud laadijaga, kuid aku isetühjeneb,
- komponendi impulsstoiteregulaatorite lülitussageduse seadistamisvõimalus, et vajadusel vähendada emiteeritava elektromagnetilise müra hulka seadme lülitusefektiivsuse arvelt ja
- katkestusteta toite tagamine USB külge- ja lahtiühendamisel.

Negatiivse temperatuurikoefitsendiga 100 kΩ nominaaltakistusega termistoriga ja takistiga koostatud pingejagur tagab laadimisvoolu katkemise olukorras, kus termistori takistus langeb 54 % tema nominaaltakistusest. Trükkplaadile valitud termistori puhul vastab see ligikaudu temperatuuriväärtusele 40°C [19]. Lisaks katkestatakse laadimine olukorras, kus termistori takistus tõuseb 317 % tema nominaaltakistusest, mis on ligilähedal temperatuuriväärtusele 0°C. [19]

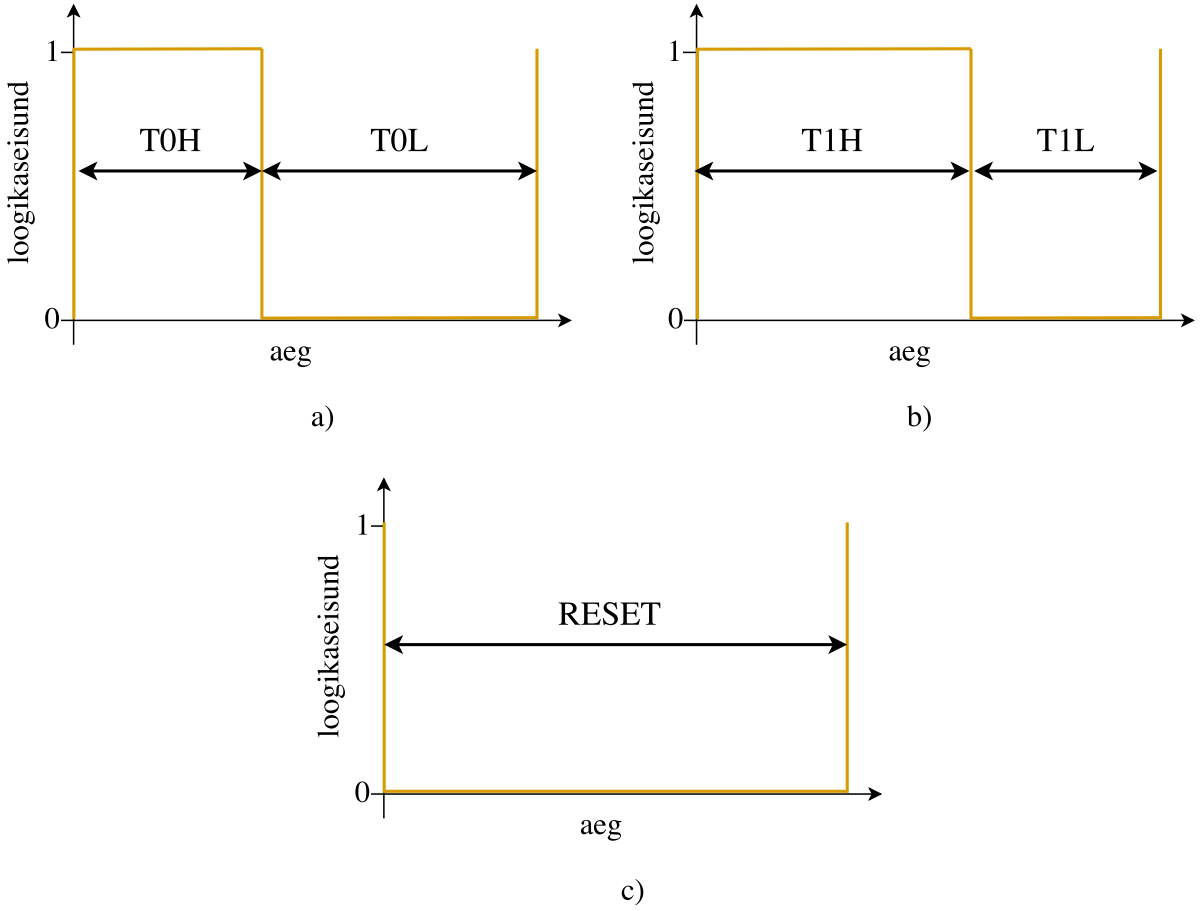
3.5 Valgusdioodindikaatorid WS2812B

WS2812B on neljaterminaliline integraallülitus, mis koosneb punarohesinisest valgusdioodist ja selle juhtelektroonikast. Komponent on käitatav 3,3 - 5 voldise toitepingega ning on kaitstud toiteterminalide valesti ühendamise vastu. Moodul WS2812B on juhitav

ühepolaarse mittenulltagastusega jadaandmesiiniga ning on disainitud nii, et komponente saaks ühendada järjestikku ühe andmesiini külge. Igat moodulit on võimalik käsitleda ühe 24-bitise pikslina, mille punase, rohelise ja sinise valguskomponendi intensiivsust on igaüht võimalik juhtida kaheksabitise märgita täisaarvu ulatuses (0 - 255). WS2812B andmeedastusprotokolli ajastusnõuded on näidatud tabelis 3.1 ning joonisel 3.3. [20]

Tabel 3.1: Andmeedastusprotokolli ajastustabel. Ühe andmebiti edastamiseks kuluva tsükli koguvältus on 1,25 µs ± 0,6 µs. [20]

Andmeedastusväljundterminali seisund ühe biti edastamiseks	**Vältus**	**Tähis**
loogilise nulli kõrge olek	400 ± 150 ns	T0H
loogilise ühe kõrge olek	800 ± 150 ns	T1H
loogilise nulli madal olek	850 ± 150 ns	T0L
loogilise ühe madal olek	450 ± 150 ns	T1L
andmeedastuse lähtestamine	$\geq$ 50 µs	RESET



Joonis 3.3: Diagrammil a on näidatud andmeedastusväljundviigu seisundi-ajagraafik ühe loogilise nulli väärtusega biti edastamiseks, diagrammil b on toodud sama graafik ühe loogilise ühe väärtusega biti edastamiseks ning diagramm c näitab andmeedastuse lähtestamise seisundi-ajagraafikut.

4 Seadme püsivara disain

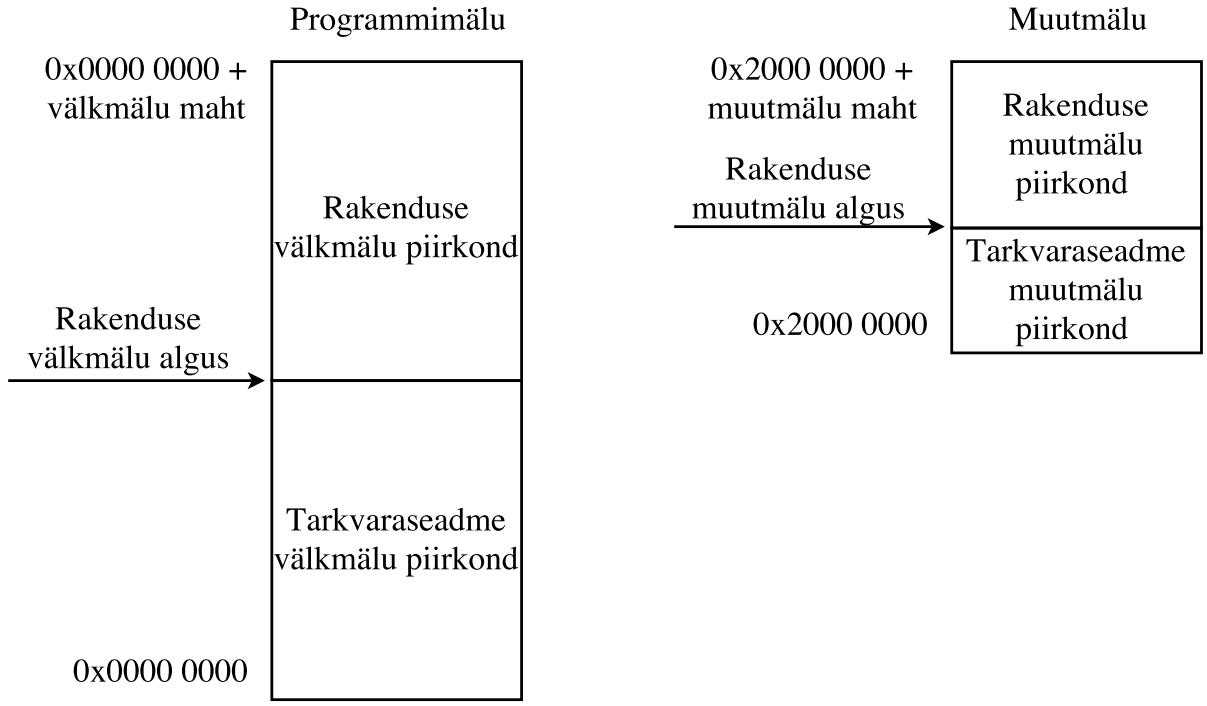
Käesolevas peatükis kirjeldatakse projekteeritud seadme mikrokontrollerile kirjutatud püsivara. Seadme mikrokontrollerile ja raadiosidemoodulile nRF51822 püsivara loomiseks kasutati *GNU* kompilaatorikogumikul põhinevat tarkvaraarendusahelat ja *GNU* silurit. Püsivara arendati programmeerimiskeeles *C*, kasutades arenduskeskkonda *Eclipse Mars 2*. Moodulile püsivara kirjutamiseks kasutati J-Link LITE CortexM [21] silurseadet koos tagconnect *TC2030-MCP* [22] ühenduspistikuga. Loodud tarkvarafailid on näidatud lisas 1.

BL600-SA-32 mooduli arendamiseks on võimalik kasutada selle tootja enda arenduskeskkonda ning programmeerimiskeelt *smart*BASIC, kuid sellest loobuti järgmistel põhjustel.

- Programmeerimiskeelel *C* põhinev püsivara on ülekantav kõigile nRF-seeria integraallülitustele ning see võimaldab vajadusel lihtsamini teha muutusi projekteeritava seadme riistvaras.
- NRF-seeria riistvarale on püsivara arendamiseks programmeerimiskeeles C hea klienditugi.
- Käesoleva töö autoril puudub varasem kokkupuude *smart*BASIC programmeerimiskeelega.

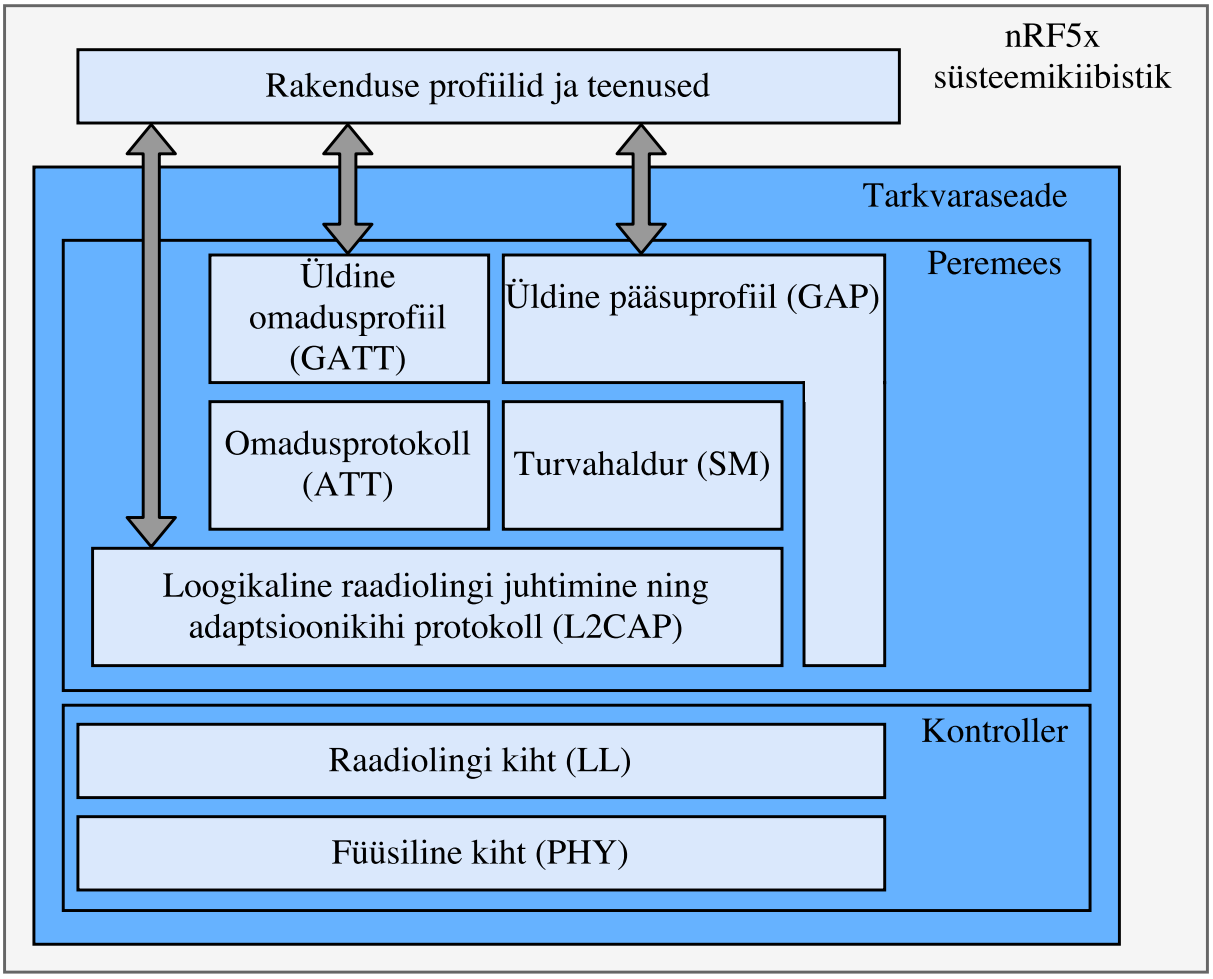
4.1 *Bluetooth Low Energy* teostus

NRF51822 raadiosidemooduli käitamiseks on selle tootja, Nordic Semiconductor, välja töötanud tarkvaraseadme (*SoftDevice*), mis on oma olemuselt eelkompileeritud ja lingitud binaarne tarkvarakiht BLE arendamise lihtsustamiseks. Projekteeritud seadmel kasutatakse tarkvaraseadme versiooni s130. Tarkvaraseadmele on toimimiseks eraldatud osa nRF51822 mäluvahenditest, mistõttu väheneb tarkvaraseadme kasutamisel programmeeritava rakenduse arendamise ressursside hulk mõnevõrra. NRF51822 mälupiirkonna jaotus on näidatud joonisel 4.1. [23]



Joonis 4.1: Tarkvaraseade kirjutatakse integraallülituse nRF51822 välkmällu enne kui rakendus. Rakenduse katse kirjutada tarkvaraseadmele eraldatud muutmäluaadressile põhjustab süsteemis veateate. [23]

BLE tarkvaraliseks realiseerimiseks projekteeritud seadmel lähtuti nRF51822 tarkvaraseadme pinuarhitektuurist tulevatest nõuetest. NRF51822 tarkvaraseadme pinuarhitektuur on näidatud joonisel 4.2. Loodav tarkvara ei edasta andmeid, mis on välja toodud BLE standardteenuste nimekirjas ning seetõttu kirjutati rakendusspetsiifiline eriteenus. Tabelis 4.1 on näidatud kirjutatud BLE-rakenduse ATT-kihi omadustabeli üldist omadusprofiili. [23]



Joonis 4.2: Rakenduse profiilide ja teenuste kiht tugineb üldise omadusprofiili (GATT) ja üldise pääsuprofiili (GAP) kihtidele. GATT-kiht põhineb omadusprotokolli (ATT) kihil, mis toimib kliendi ja serveri seosel. Serverina hoitakse ATT-kihi omadustabelis näiteks anduritelt kogutud andmeid. GATT-kihi ülesanne on omadustabelis olevad kirjed grupeerida kindla struktuuri alusel. Loogikaline raadiolingi juhtimise ning adaptsioonikihi protokoll (L2CAP) haldab samal raadiolingi kihil (LL) olevate teenuste ning protokollide ümberlülitamist. [23]

TABEL 4.1: Arendatud teenuse korrastatud omadustabel. Omaduse pide määrab unikaalselt serveris oleva omaduse, millele klient saab kirjutamis- või lugemisprotsessis viidata. Omadustüüp kirjeldab globaalset identifikaatorit, millega on võimalik igat omaduse tüüpi määrata. Omaduse pääsuluba määrab omadusega suhtlemise piirangud. Omaduse väärtus hoiustab erinevate teenuste globaalseid identifikaatoreid ning anduri mõõtetulemusi.

Teenus	**Pide**	**Omadusetüüp, UUID**	**Omaduse pääsuluba**	**Omaduse väärtus**
Teenuse deklaratsioon	0x000X	**Teenuse karakteristik**, standardne teenuse UUID 0x2800	Kirjutuskaitstud, autentimiseta, autoriseerimiseta	Teenuse UUID 0x0000food-1212-efde-1523-785fef13d123
Karakteristiku deklaratsioon	0x000X	**Karakteristiku deklaratsioon**, standardne karakteristiku UUID 0x2803	Kirjutuskaitstud, autentimiseta, autoriseerimiseta	Karakteristiku UUID 0x0000beef-1212-efde-1523-785fef13d123
Karakteristiku väärtuse deklaratsioon	0x000X	**Karakteristik**, UUID 0x0000beef-1212-efde-1523-785fef13d123	Kirjutuskaitstud, autentimiseta, autoriseerimiseta	Anduri mõõtetulemus ning akupinge (17 uint16_t-tüüpi elemendiga massiiv)
Deskriptori deklaratsioon	0x000X	**Kliendi karakteristikusätete deskriptor**, standardne teenuse UUID 0x2902	Kirjutuskaitseta, autentimiseta, autoriseerimiseta	Teavitus käivitatud 0x00-XX

Kõigi andmete ja teenuse käitamiseks vajalike parameetrite hoidmiseks kasutatakse põhiprogrammis `main.c` andmestruktuuri `ble_os_t` tüüpi muutuja `m_our_service` deklareerimisega. Konstrueeritav teenus realiseeritakse funktsioonis `services_init`, kus kutsutakse välja funktsioon `our_service_init()`, mille parameeter on viit `&m_our_service`. Funktsioonis `our_service_init` ↪ `()` defineeritakse vajalikud globaalsed identifikaatorid (UUID) ning initsialiseeritakse varasemalt defineeritud teenus funktsiooniga `sd_ble_gatts_service_add()`, mis kasutab kolme parameetrit, kus esimene määrab teenuse prioriteeditüübi, teine on viit teenuse globaalsele identifikaatorile ning kolmas on viit 16-bitisele sõnele, kus hoiustatakse määratud pide:

```
err_code = sd_ble_gatts_service_add(BLE_GATTS_SRVC_TYPE_PRIMARY,
                                    &service_uuid,
                                    &p_our_service->service_handle);
APP_ERROR_CHECK(err_code);
```

BLE tutvustusfunktsionaalsus initsialiseeritakse funktsiooni `advertising_init()` väljakutsega. Funktsioonis deklareeritakse andmestruktuuri `ble_advdata_t` tüüpi muutujad `advdata` ning `srdata`, mis sisaldavad sätteid tutvustatavate andmete defineerimiseks. Andmestruktuuri `ble_adv_modes_config_t` tüüpi muutuja `options` deklareerimisega seatakse tutvustusrežiimi parameetrid. Tutvustusrežiimil edastatavas andmepaketis edastatava kasuliku informatsiooni hulk on maksimaalselt 31 baiti. Funktsioonis `advertising_init()` määratakse tutvustava seadme nimi ja tutvustamisrežiimis edastatavate andmepakettide vaheline intervall:

```
    ble_advdata_t advdata;

    // Build advertising data struct to pass into ble_advertising_init().
    memset(&advdata, 0, sizeof(advdata));

    advdata.name_type               = BLE_ADVDATA_FULL_NAME;
    advdata.flags                   = BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE;

    ble_advdata_t srdata;
    memset(&srdata, 0, sizeof(srdata));
    srdata.uuids_complete.uuid_cnt = sizeof(m_adv_uuids) / sizeof(m_adv_uuids[0]);
    srdata.uuids_complete.p_uuids = m_adv_uuids;
```

```
    ble_adv_modes_config_t options = {0};
    options.ble_adv_fast_enabled  = BLE_ADV_FAST_ENABLED;
    options.ble_adv_fast_interval = APP_ADV_INTERVAL;
    options.ble_adv_fast_timeout  = APP_ADV_TIMEOUT_IN_SECONDS;
```

ATT-kihis olevate andmete struktureerimiseks kirjutati teenusele karakteristiku lisamiseks funktsioon `our_char_add()`. Funktsiooni kolm olulisimat muutujat on:

- `ble_gatts_attr_md_t`-tüüpi muutuja `attr_md`, mis kirjeldab omaduse metaandmeid. Andmestruktuur sisaldab pääsu- ja autoriseerimise andmeid ning määrab karakteristiku väärtuse fikseeritud pikkusega väärtuseks ning määrab karakteristiku asukoha mälus:

  ```
      ble_gatts_attr_md_t attr_md;
      memset(&attr_md, 0, sizeof(attr_md));
      attr_md.vloc        = BLE_GATTS_VLOC_STACK;;
  ```

- `ble_gatts_char_md_t`-tüüpi muutuja `char_md`, mis kirjeldab karakteristiku metaandmeid. Andmestruktuur hoiustab karakteristiku väärtuse sätteid ning deskriptorite metaandmeid:

```
        ble_gatts_char_md_t char_md;
        memset(&char_md, 0, sizeof(char_md));
        char_md.char_props.read = 1;
        char_md.char_props.write = 1;
```

- `le_gatts_attr_t`-tüüpi muutuja `attr_char_value`, mis kirjeldab karakteristiku väärtuse omadust. Andmestruktuur hoiustab karekteristiku väärtust, selle maksimaalset pikkust ning globaalset identifikaatorit:

```
    ble_gatts_attr_t    attr_char_value;
    memset(&attr_char_value, 0, sizeof(attr_char_value));
    attr_char_value.p_uuid      = &char_uuid;
    attr_char_value.p_attr_md   = &attr_md;

    attr_char_value.max_len     = 17;
    attr_char_value.init_len    = 17;
    uint8_t value[4]            = {0x12,0x34,0x56,0x78};
    attr_char_value.p_value     = value;
```

Lisaks määratakse funktsioonis `our_char_add()` karakteristikule globaalne identifikaator. Karakteristiku baasidentifikaator on sama, mis teenuse baasidentifikaator ning karakteristiku enda identifikaator on 16-bitine unikaalne väärtus. Karakteristiku pääsusätete määramiseks kasutatakse makrot `BLE_GAP_CONN_SEC_MODE_SET_OPEN()`:

```
        ble_uuid_t          char_uuid;
        ble_uuid128_t       base_uuid = BLE_UUID_OUR_BASE_UUID;
        char_uuid.uuid      = BLE_UUID_OUR_CHARACTERISTC_UUID;
        err_code = sd_ble_uuid_vs_add(&base_uuid, &char_uuid.type);

    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&attr_md.read_perm);
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&attr_md.write_perm);
```

Kliendi karakteristikusätete deskriptori (CCCD) lisamiseks funktsioonile `our_char_add()` deklareeritakse konfiguratsioonisätete hoidmiseks metaandmete struktuur ning määratakse pääsusätted. Lisaks määratakse deskriptori hoiustamiseks tarkvaraseadme juhitud mälupiirkond ning hoiustatakse CCCD metaandmede struktuur karakteristiku metaandmete struktuuris. Lõpetuseks lubatakse teavituste edastamine:

```
    ble_gatts_attr_md_t cccd_md;
    memset(&cccd_md, 0, sizeof(cccd_md));
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&cccd_md.read_perm);
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&cccd_md.write_perm);
    cccd_md.vloc                = BLE_GATTS_VLOC_STACK;
    char_md.p_cccd_md           = &cccd_md;
    char_md.char_props.notify   = 1;
```

Andmete edastamine toimub funktsioonis `characteristic_update()`. Funktsioonis kontrollitakse, kas server on kliendiga ühenduses ning deklareeritakse `ble_gatts_hvx_params_t`-tüüpi muutuja `hvx_params`, kus hoitakse teavituse sätete parameetreid:

- pidet, mis viitab kasutatava karakteristiku väärtusele,
- pide tüüpi, mis määrab, et edastatakse teavitus,
- andmete nihkeväärtust, mis võimaldab edastada andmemassiivi sees olevaid andmeid,
- väärtust, mis määrab edastatavate baitide hulga,
- andmeviita, mis osutab edastatavatele andmetele.

Andmestruktuur `hvx_params` ning konkreetse kliendi ja serveri ühenduse pide antakse funktsiooni `sd_ble_gatts_hvx()` parameetriteks ning teavitus edastatakse.

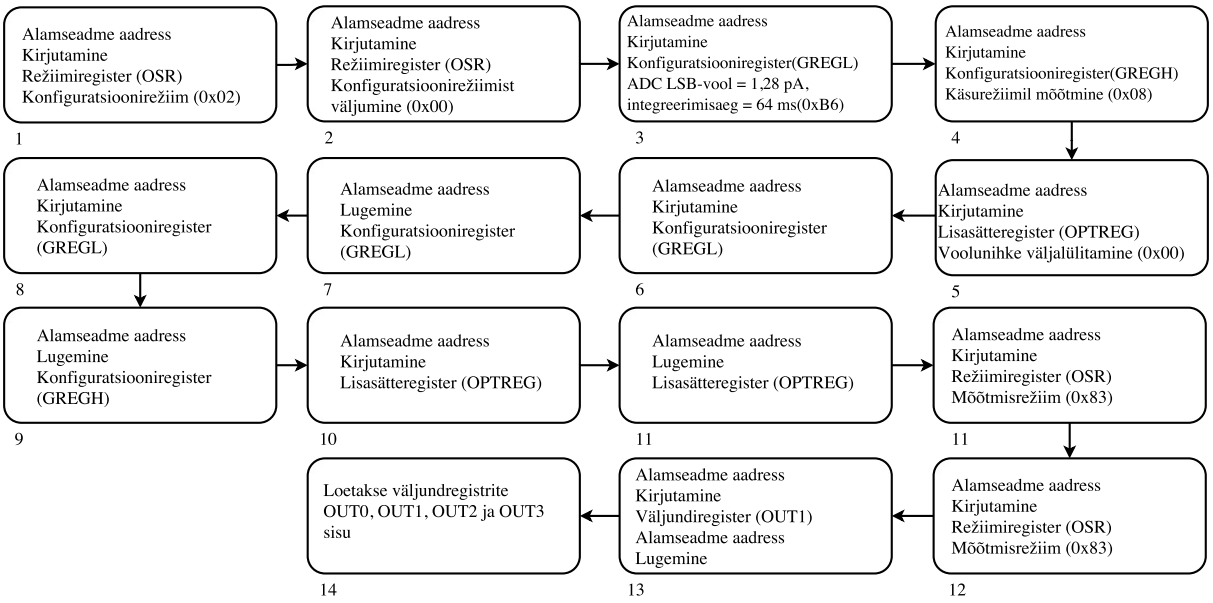
```
if (p_our_service->conn_handle != BLE_CONN_HANDLE_INVALID)
{
        uint16_t               len = 17;
        ble_gatts_hvx_params_t hvx_params;
        memset(&hvx_params, 0, sizeof(hvx_params));

        hvx_params.handle = p_our_service->char_handles.value_handle;
        hvx_params.type   = BLE_GATT_HVX_NOTIFICATION;
        hvx_params.offset = 0;
        hvx_params.p_len  = &len;
        hvx_params.p_data = data;

        err_code = sd_ble_gatts_hvx(p_our_service->conn_handle, &hvx_params);
```

4.2 MCDC04 juhtimine

Analoog-digitaalmuundurit MCDC04 juhitakse üle tarkvaraliselt realiseeritud I^2C-protokolli. Ühe alamseadme juhtimisprotokolli skeem on toodud joonisel 4.3.



Joonis 4.3: Ühekordse seadistamis- ning mõõtmisprotseduuri läbiviimiseks kirjutatakse juhtimisregistritesse soovitavad väärtused ning seadistamise õnnestumist kontrollitakse juhtimisregistrite väärtuse lugemisega. Pärast mõõterežiimi sisenemist on võimalik teostada mõõtmised, mille tulemused kirjutatakse väljundregistritesse. Väljundregistrite OUT1-OUT3 suurus on 16 bitti. Kõikide juhtimisregistrite suurus on 8 bitti.

MCDC04 mõõtmisi teostatakse seadme käsurežiimis ning mõõtmiskäsu edastamine ja andmete lugemine toimuvad mikrokontrolleri taimeri katkestuse ajal kutsutavas funktsioonis `send_bulk_data()`. Taimeri katkestuse periood on 500 ms. Mõõtetulemused kirjutatakse kuueteistkümnesse `uint16_t`-tüüpi andmevälja ning edastatakse seejärel funktsiooniga `characteristic_update(&m_our_service, output_data)`:

```
void send_bulk_data(void) {

    // Update characteristic value.
        uint32_t        err_code = 0;
    measure();
    read();
    convert_data();
    adc_1();
    output_data[16] = adc_result;

                characteristic_update(&m_our_service, output_data);
                if (//(err_code != NRF_SUCCESS) &&
                                        (err_code != NRF_ERROR_INVALID_STATE) &&
                                        (err_code != BLE_ERROR_NO_TX_PACKETS)// &&
                                        //(err_code != BLE_ERROR_GATTS_SYS_ATTR_MISSING)
                                        ){}

}
```

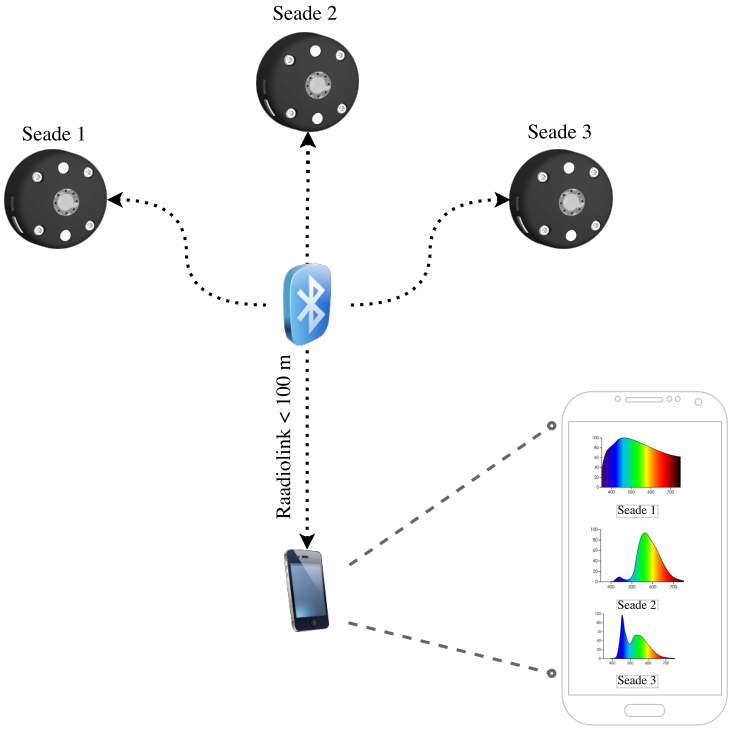
4.3 WS2812B juhtimine

Valgusdioodmooduli WS2812B juhtimiseks kasutatakse teeki Neopixel, mis implementeerib mooduli käitamiseks vajalike täpsete ajastusnõute täitmiseks assemblerkeeles kirjutatud *NOP*-käske. BLE rakendamisel võtab tarkvaraseade protsessori juhtimise üle olukorras, kus on vaja saatele minna – selliste ajakriitiliste protsesside katkestuste prioriteet on alati kõrgem, kui mistahes rakenduste oma [23]. WS2812B juhtimiseks on vaja, et andmed edastataks korraga ning õige ajastusega ning selleks, et kõrgema prioriteediga protsessid valgusdioodmoodulitele andmete edastamist ei katkestaks, kasutatakse raadioseansi teavitust. Raadioseansi teavitus on tarkvaraline katkestus, mis teavitab rakendust lähenevast raadioseansist määratud ajal enne seansi algust. Raadioseansi teavituse määramiseks kasutakse funktsiooni `radio_notification_init(uint32_t irq_priority, uint8_t notification_type, uint8_t notification_distance)`. Raadioseansi teavituse tarkvaralise katkestuse pides teostatakse valgusdioodidmoodulitele signaalide edastamine:

```
void SWI1_IRQHandler(bool radio_evt)
{
    if (radio_evt)
    {
        if (led_status == 1)
        {
                neopixel_set_color_and_show(&m_strip, led_to_enable, red, green, blue);
                led_status = 0;
        }
    }
}
```

5 Lõputöö perspektiiv

Käesoleva lõputöö raames konstrueeritud seadme elektroonika on funktsioneeriv ning kirjutatud püsivara võimaldab edukalt edastada andmeid. Valitud anduri omaduste tõttu on edastatud andmete põhjal võimalik lisaks valgustatusele ning valguse värvitemperatuurile matemaatilisi mudeleid kasutades konstrueerida ka valguse spektraalse jaotuse lähendus. Selleks on vaja katseliselt hinnata seadme spektraalset tundlikkust. Võimekus teostada valgusspektri mõõtmisi ning mõõteseadme sidestatavus nutiseadmega, teeb projekteeritud prototüübi disaini uuenduslikuks. Vastava nutirakenduse loomine võimaldaks katsetada korraga enam kui ühe mõõteseadme ühendamist nutiseadmega, mis annaks kasutajale reaalajas tagasisidet mitmes ruumipunktis valitsevate valgusomaduste üle. Süsteemi kontseptsioonskeemi on näidatud joonisel 5.1.



Joonis 5.1: *Bluetooth Low Energy* spetsifikatsioonid ning nutiseadmete võimekus lubab nutiseadmega ühendada rohkem kui ühe mõõtepunkti korraga. Mõõtepunktides mõõdetud andmeid edastataks reaalajas ning arvutatud tulemused kuvataks kasutaja nutiseadme ekraanil.

Kokkuvõte

Valguse omaduste teadmine mingis ruumipunktis on oluline mitmete elukutsete esindajatele. Valdavalt kasutatakse selleks akutoitel põhinevaid käsimõõteseadmeid, mille valgustundlik element asub seadme küljes ning võimaldab mõõta valguse omadusi ajas korraga vaid ühes ruumipunktis. Tänapäevane raadiosidetehnoloogia, kõrge integratsiooniastmega elektroonikakomponendid ja nutiseadmete ulatuslik levik võimaldavad valguse omaduste mõõtmiseks välja arendada uuenduslikke lahendusi.

Käesoleva lõputöö eesmärk oli projekteerida ning konstrueerida elektrooniline seade, mis oleks võimeline muundama valgustundlikult tajurilt saadud signaali digitaalseks ning mis edastaks need üle *Bluetooth Low Energy* raadiopersonaalvõrgu tehnoloogia nii, et neid oleks võimalik nutiseadmega lugeda. Nimetatud funktsionaalsuse saavutamiseks kirjutati seadme mikrokontrollerile püsivara, mis tagab elektrooniliste komponentide juhtimise ning raadiopersonaalvõrgu funktsioneerimise. Lisaks projekteeriti elektroonilise seadme ümbritsemiseks mehaanilise korpuse osised ning konstrueeriti esmane korpuse prototüüp.

Käesolevas töös tehti ülevaade valikust radiomeetrilistest ja fotomeetrilistest füüsikalistest suurustest ning kirjeldati värvustemperatuuri teoreetilisi aluseid. Lõputöös käsitletava tajuri tööpõhimõtete tausta selgitamiseks uuriti lühidalt optiliste filtrite olemust ning pooljuhtfotodetektorite käitumist. Lisaks tehti ülevaade projekteeritud seadme poolt rakendatud raadiopersonaalvõrgu tehnoloogia spetsifikatsioonidest. Lõputöös kirjeldatakse konstrueeritud seadme kõiki alamsüsteeme – implementeeritud elektroonikat, mehaanilist disaini ning püsivara.

Kasutatud kirjandus

[1] Tööeluportaal. Valgustus, 2016. URL http://www.tooelu.ee/et/Tooandjale/Tookeskkond/Tookeskkonna-ohutegurid/Fyysikalised-ohutegurid/valgustus. (07.05.2017).

[2] T. Ang. *Digital Photographer's Handbook*. Dorling Kindersley, 6th edition, 2016.

[3] Lumulabs. Lumu, 2017. URL https://lu.mu/. (07.05.2017).

[4] A. J. Wilt. Cine Meter II for iPhone, 2017. URL https://www.adamwilt.com/cinemeterii/. (07.05.2017).

[5] D. Baer. How Many People Own Smartphones Around the World, 2016. URL http://www.businessinsider.com/how-many-people-own-smartphones-around-the-world-2016-2. (07.05.2017).

[6] The Statistics Portal. Number of smartphone users worldwide from 2014 to 2020, 2014. URL https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. (07.05.2017).

[7] D. Malacara. *Color Vision and Colorimetry: Theory and Applications*. SPIE Press, 2nd edition, 2011.

[8] P. Hobbs. *Building Electro-Optical Systems – Making It All Work*. Wiley, 2nd edition, 2009.

[9] R. Paschotta. Article on 'optical filters' in the encyclopedia of laser physics and technology, 2008. URL https://www.rp-photonics.com/optical_filters.html?s=ak. (05.04.2017).

[10] R. Nave. Interference filters, 2008. URL http://hyperphysics.phy-astr.gsu.edu/hbase/phyopt/intfilt.html. (05.04.2017).

[11] Edmund Optics. Imagining Optics - A Technical Resource for Imaging Solutions, 2017.

[12] S. B. Alexander. *Optical Communication Receiver Design*. SPIE Press, 1997.

[13] S. Rajbhandari Z. Ghassemlooy, W. Popoola. *Optical Wireless Communications – System and Channel Modelling with MATLAB*. CRC Press, 2012.

[14] Bluetooth Special Interest Group. Bluetooth Low Energy. `https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works/low-energy`, 2017. (07.05.2017).

[15] *Bluetooth Core Specification*. Bluetooth Special Interest Group, 2016. Rev. 5.0.

[16] Technopedia. Time Division Duplex (TDD), 2017. URL `https://www.techopedia.com/definition/27019/time-division-duplex-tdd`. (09.05.2017).

[17] *6-channel Multiple Color Sensor*. ams Sensors Germany GmbH, 1 2016. Rev. 6.4.

[18] *16 bit 4-channel analog-to-digital converter (ADC) with I2C control/output*. ams Sensors Germany GmbH, 2016. Rev. 4.3.

[19] *Computation spreadsheet for VISHAY BC components of reference NTCS0603 series SMD0603*. Vishay, 2011.

[20] *Intelligent control LED integrated light source*. Worldsemi. URL `http://www.seeedstudio.com/document/pdf/WS2812B%20Datasheet.pdf`.

[21] SEGGER Microcontroller. J-Link LITE CortexM-9 / J-Link LITE CortexM-19, 2017. URL `https://www.segger.com/jlink-lite-cortexm.html`. (10.05.2017).

[22] Tag-Connect. TC2030-MCP 6-Pin Cable with RJ12 Modular Plug, 2017. URL `http://www.tag-connect.com/contact`. (10.05.2017).

[23] *SoftDevice Specification S130 SoftDevice v2.0*. Nordic Semiconductor, 2016. Rev. 2.0.

WIRELESS ACCESSORY FOR A SMARTPHONE TO MEASURE ILLUMINANCE AND COLOUR TEMPERATURE

Summary

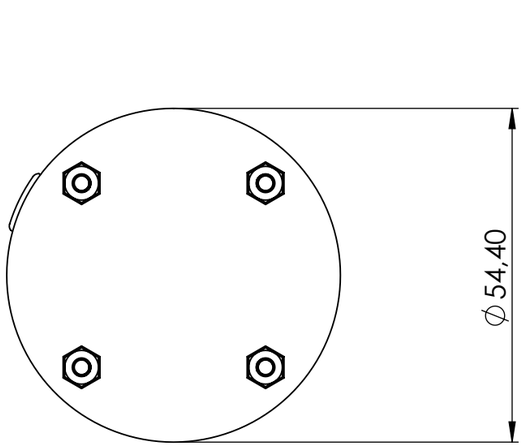
Being aware of the lighting conditions in the surrounding environment is important for various specialists. Currently, battery-operated hand-held devices are mainly used for measuring the properties of light. However, these devices are limited to taking measurements in a single point in space at a given time. Modern wireless communication technologies, highly integrated microchips and the increasing usage of smartphones and tablet computers enable the development of innovative solutions for measuring the properties of light.

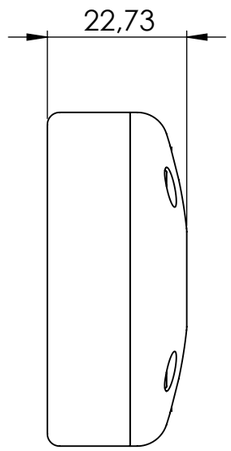
The aim of this thesis was to design and construct an electronic device that is able to convert the output of a multiple colour sensor to a digital value and transmit the gathered data to a smartphone using the Bluetooth Low Energy protocol. In order to achieve these goals, firmware for controlling the electronics and setting up the Bluetooth Low Enery protocol was produced. Furthermore, a case for covering the electronics was designed and a preliminary mechanical prototype was constructed.

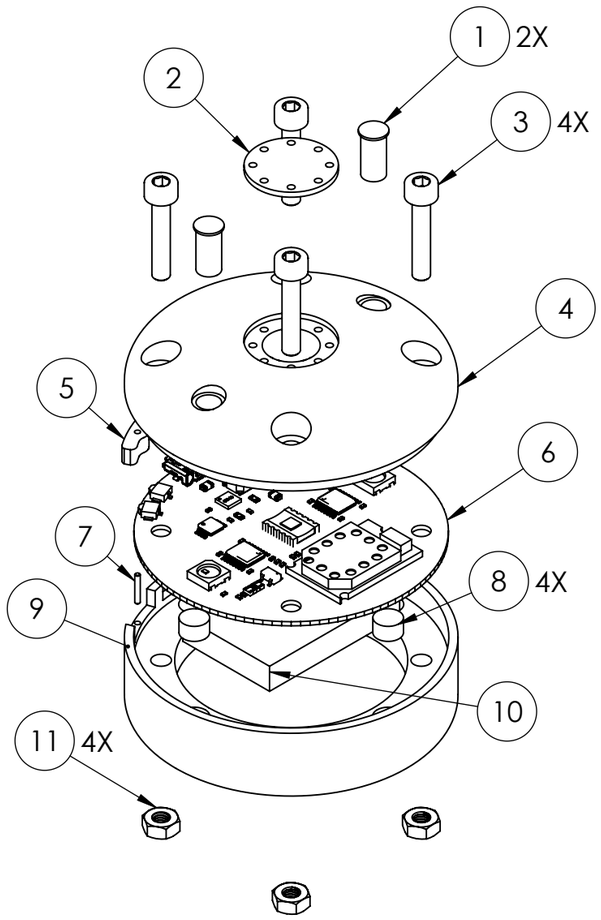
This thesis gives an overview of selected radiometric and photometric units, as well as the principles of colour temperature. In order to better comprehend the sensor used in the constructed device, the fundamentals of optical filters and semiconductor photodetectors are described. In addition, an overview of the specifications of Bluetooth Low Energy is given. Lastly, all the subsystems of the constructed device are described.

LISAD

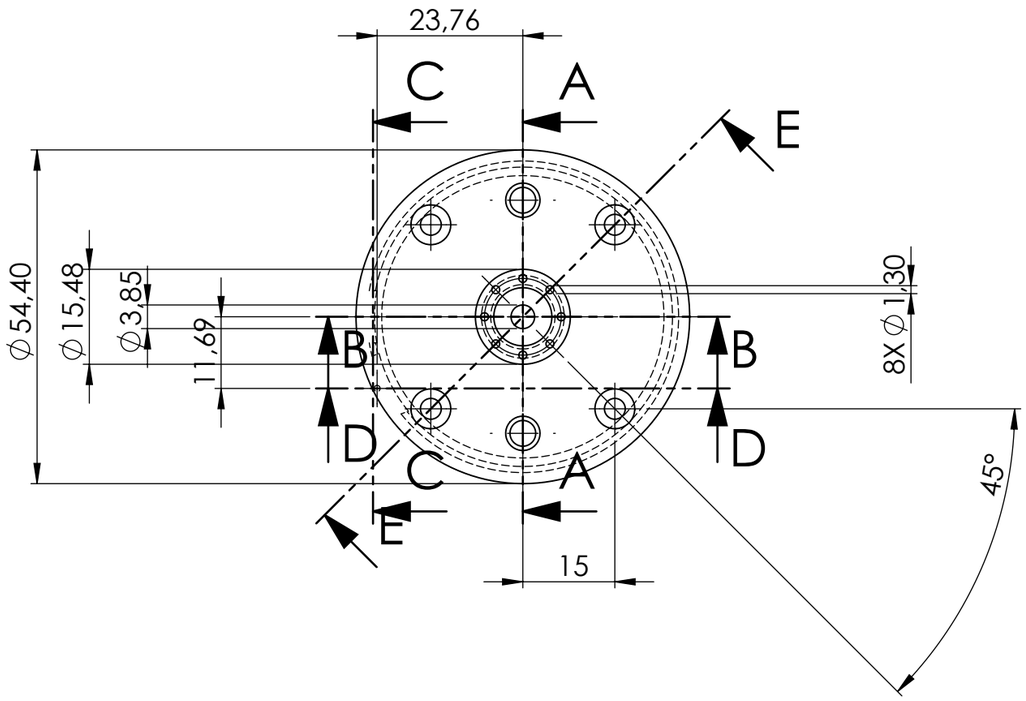
LISA A - Seadme korpus

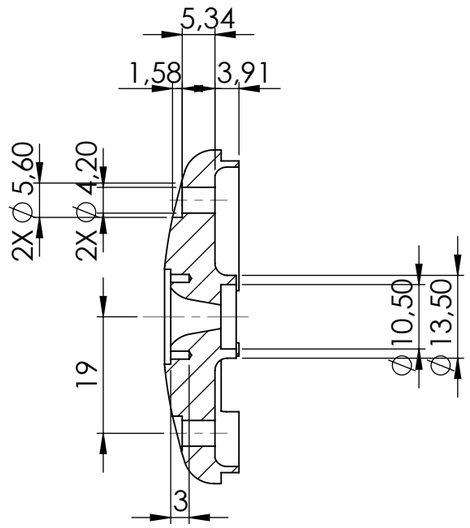




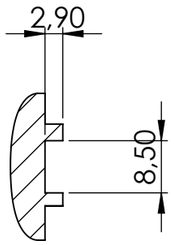


1		Valgustoru MENTOR 1293.1100		2	
2		Kaitseklaas	TN 17/120463 A 01 02 D	1	
3		DIN 7984-M3x15-8.8 A2B		4	
4		Korpuse esikaas	TN 17/120463 A 01 04 D	1	
5		Korpuse lüliti	TN 17/120463 A 01 05 D	1	
6		Trükkplaat		1	
7		Tüübel 1x6 mm		1	
8		Neodüümmagnet 5x3 mm		4	
9		Korpuse tagakaas	TN 17/120463 A 01 09 D	1	
10		Liitiumioon-aku L502030		1	
11		DIN 934-M3-8 A2B		4	
Osa	Väli	Nimetus, materjal	Tähis	Hulk	Märkus
		Materjal:	Näitamata piirhälbed: ISO 2768	Mass:	Mõõt:
Teostas		Jan Bogdanov	Nimetus: Seadme koost		
Kontrollis		Tanel Ainla			
Kinnitas		Tanel Ainla			
EMU TS - TN			Leht: 1	Tähis: TN 17/120463 A 01 01 K	

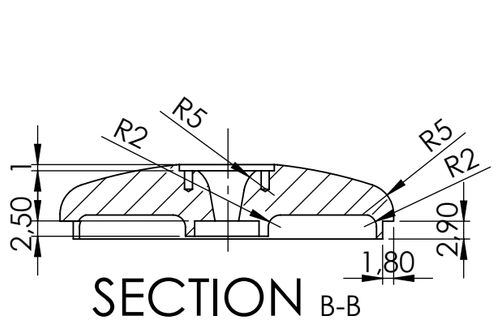




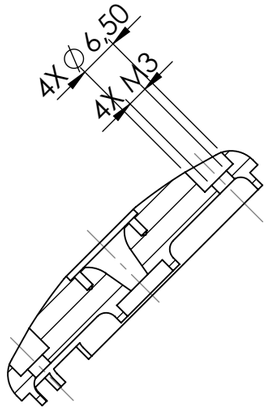
SECTION A-A



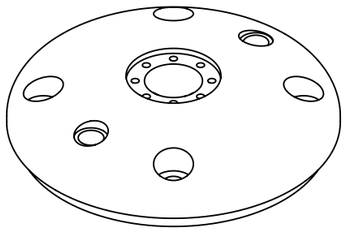
SECTION C-C

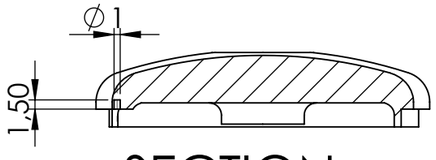


SECTION B-B



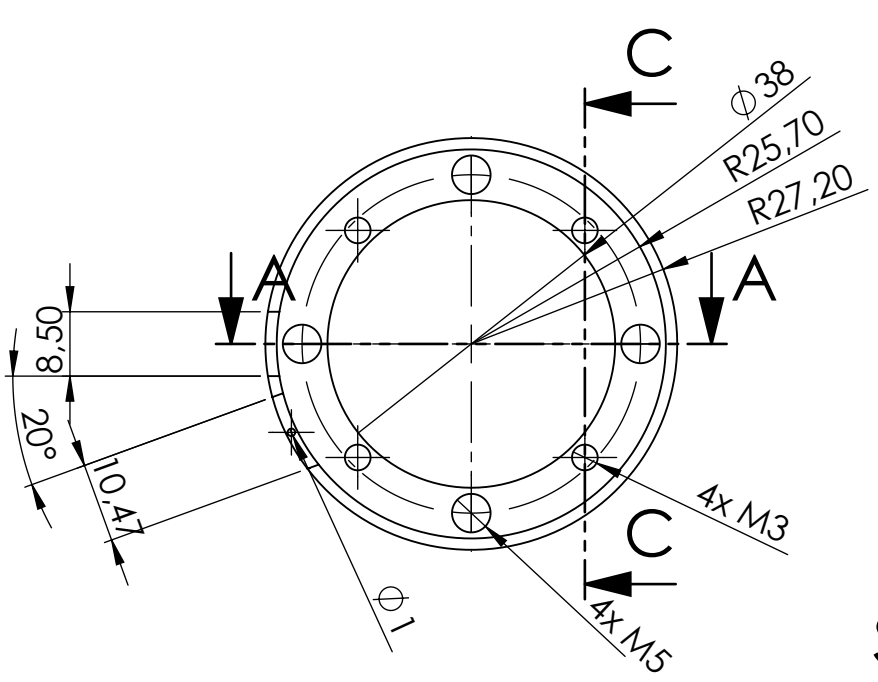
SECTION E-E

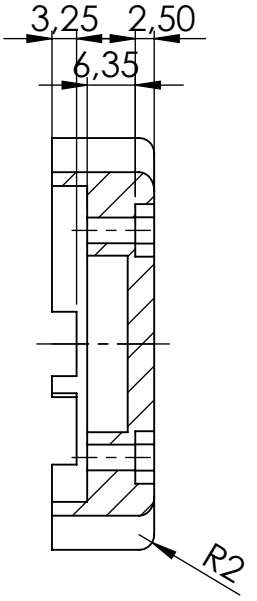




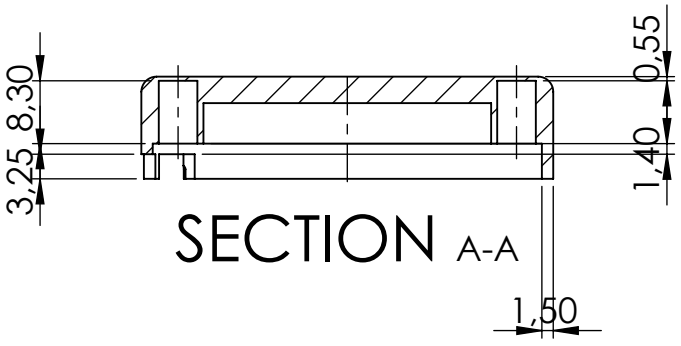
SECTION D-D

	Materjal: ABS	Näitamata piirhälbed: ISO 2768	Mass:	Mõõt: 1:1
Teostas	Jan Bogdanov	Nimetus: Korpuse esikaas		
Kontrollis	Tanel Ainla			
Kinnitas	Tanel Ainla			
EMU TS - TN		Leht: 1	Tähis: TN 17/120463 A 01 04 D	

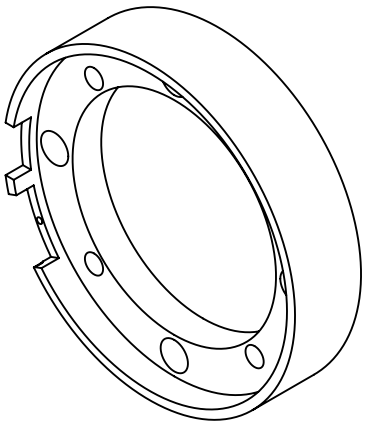




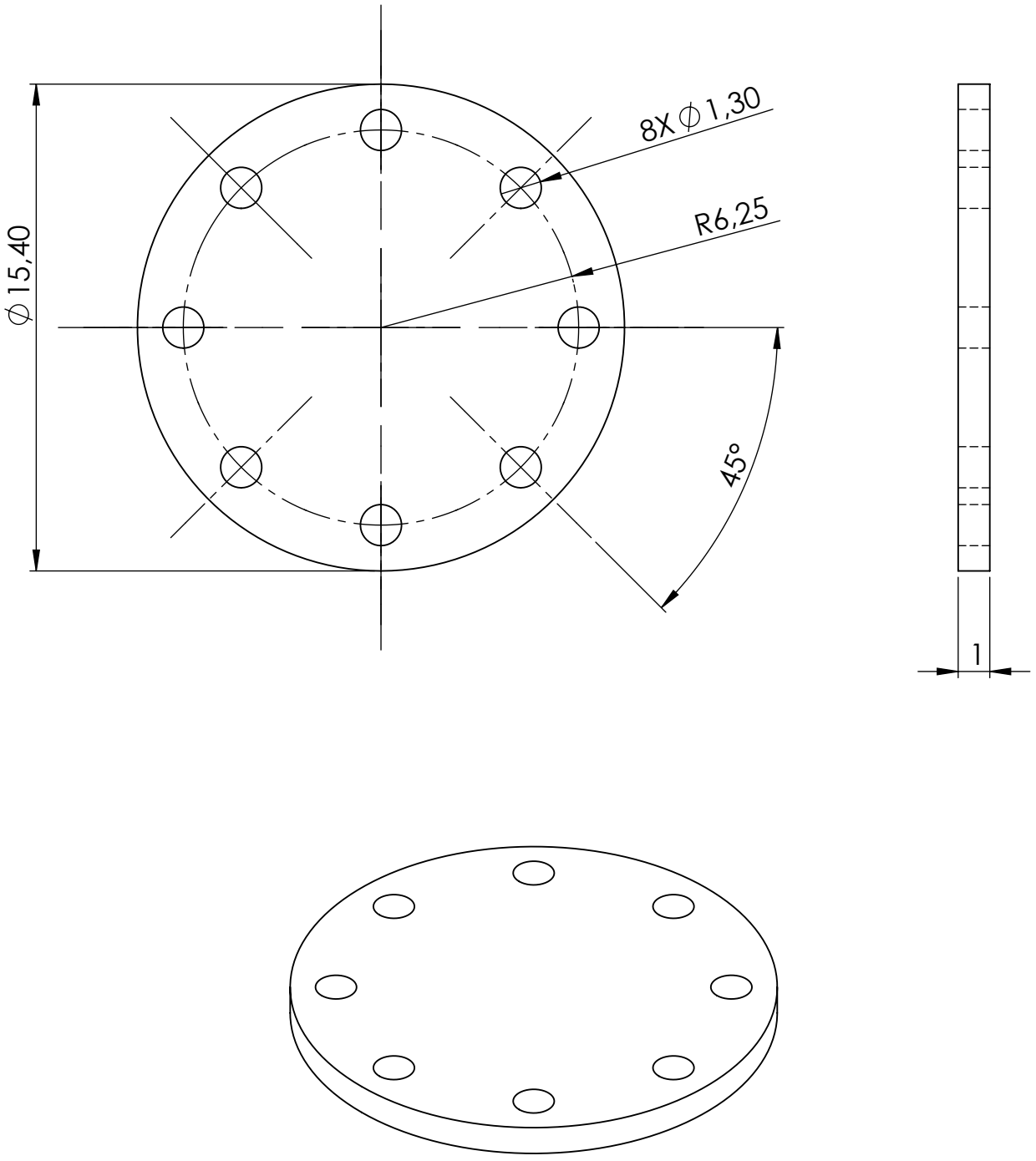
SECTION C-C



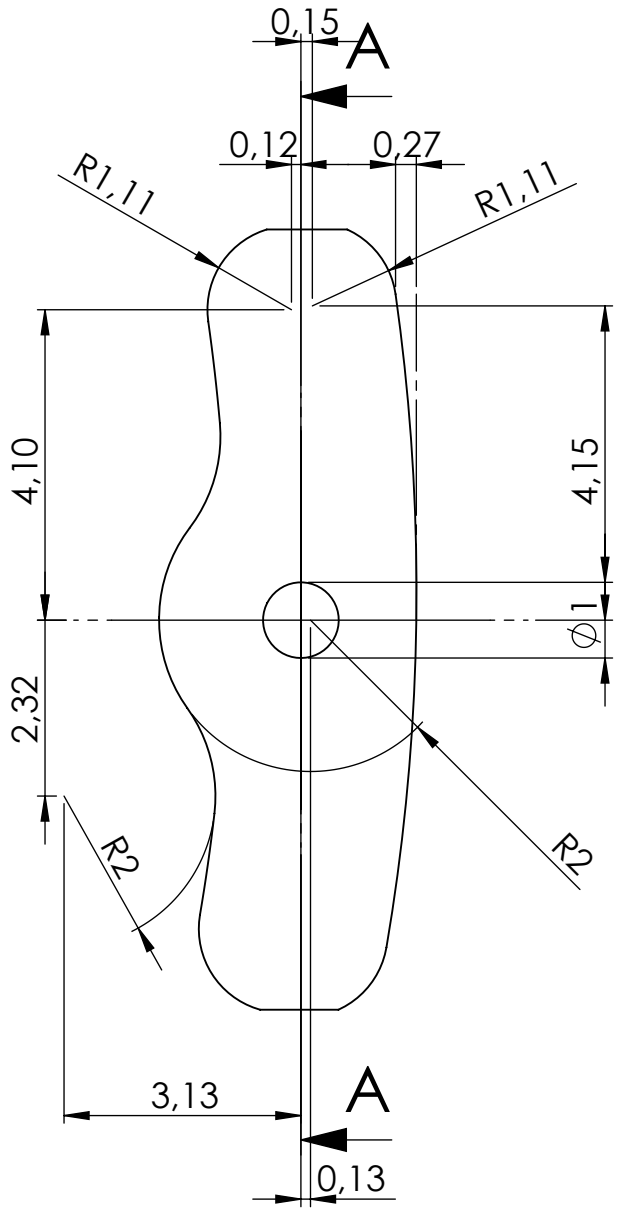
SECTION A-A

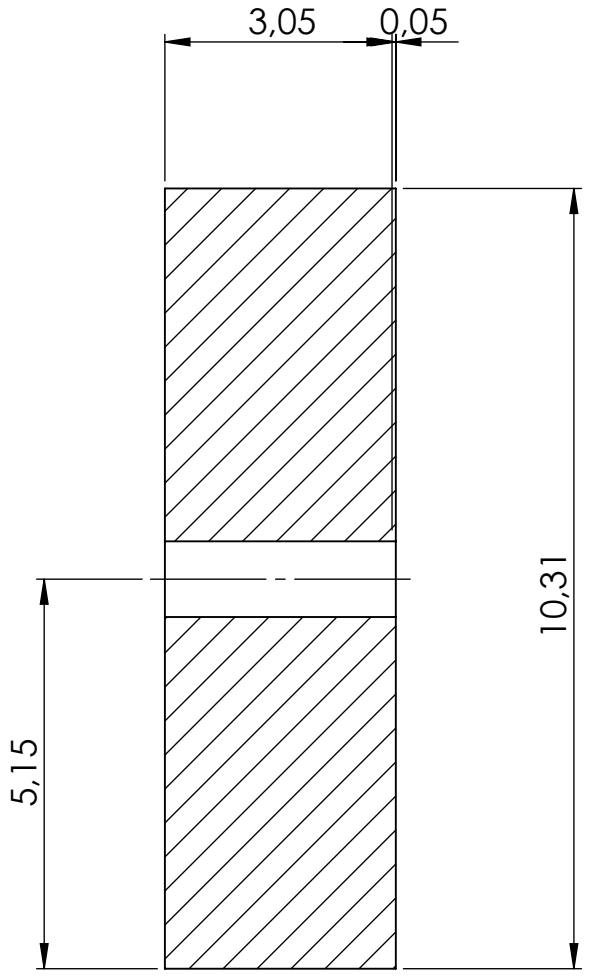


Materjal: ABS		Näitamata piirhälbed: ISO 2768	Mass:	Mõõt: 1:1
Teostas	Jan Bogdanov	Nimetus: Korpuse tagakaas		
Kontrollis	Tanel Ainla			
Kinnitas	Tanel Ainla			
EMU TS - TN		Leht: 1	Tähis: TN 17/120463 A 01 09 D	



	Materjal:	Näitamata piirhälbed: ISO 2768	Mass:	Mõõt: 1:5
Teostas	Jan Bogdanov	Nimetus: Kaitseklaas		
Kontrollis	Tanel Ainla			
Kinnitas	Tanel Ainla			
EMU TS - TN		Leht: 1	Tähis: TN 17/120463 A 01 02 D	

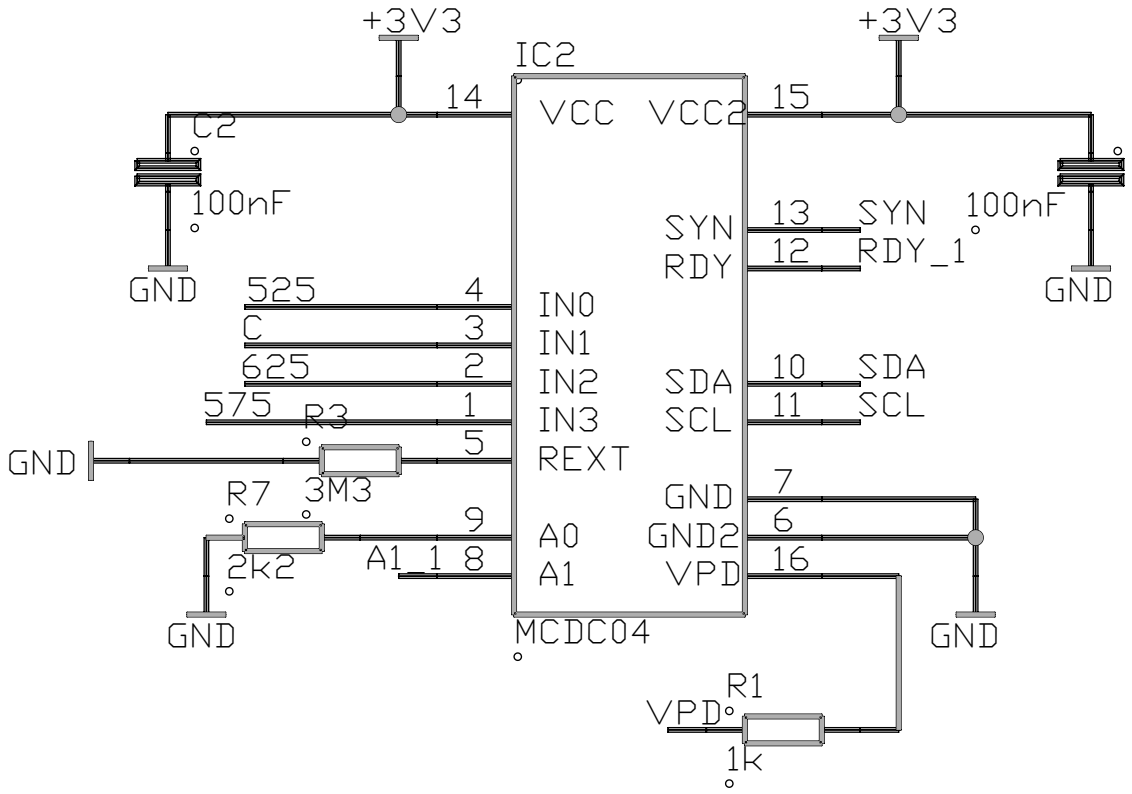




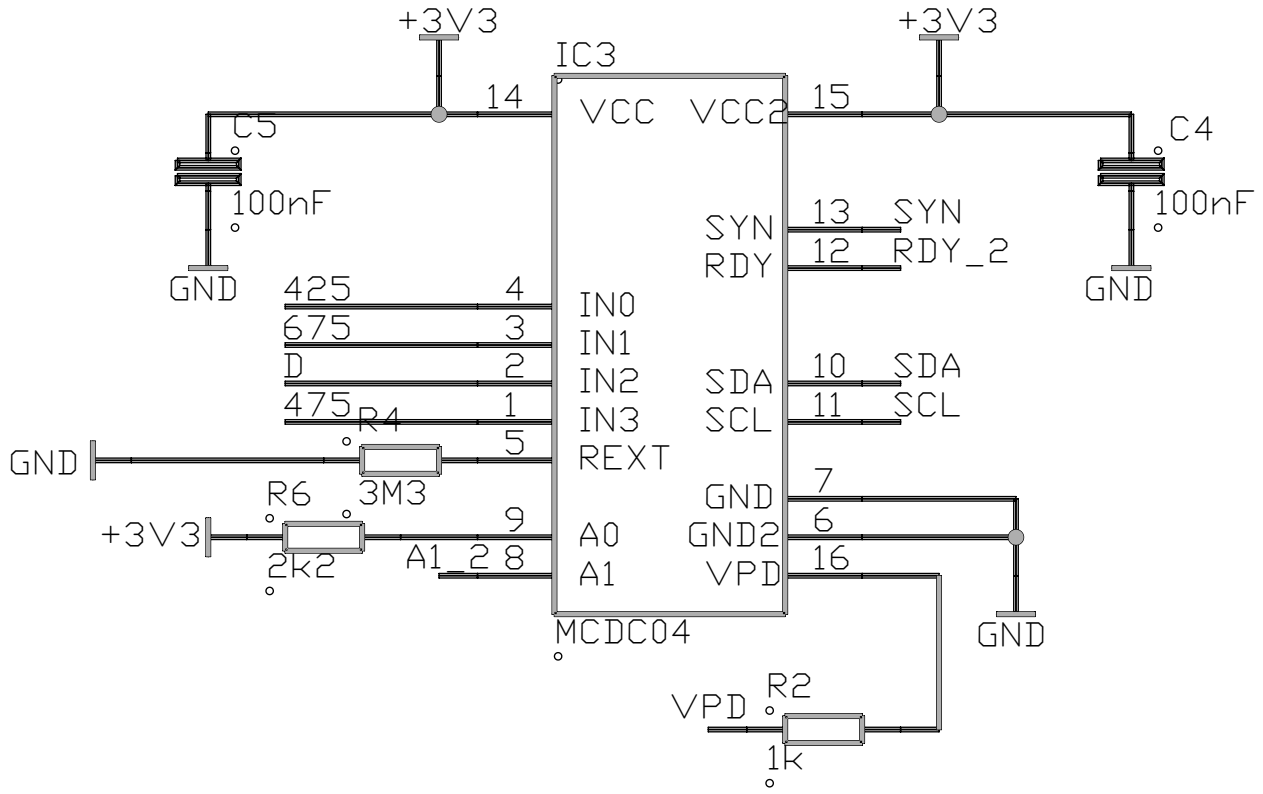
SECTION A-A

	Materjal: ABS	Näitamata piirhälbed: ISO 2768	Mass:	Mõõt: 10:1
Teostas	Jan Bogdanov	Nimetus: Korpuse lüliti		
Kontrollis	Tanel Ainla			
Kinnitas	Tanel Ainla			
EMU TS - TN		Leht: 1	Tähis: TN 17/120463 A 01 05 D	

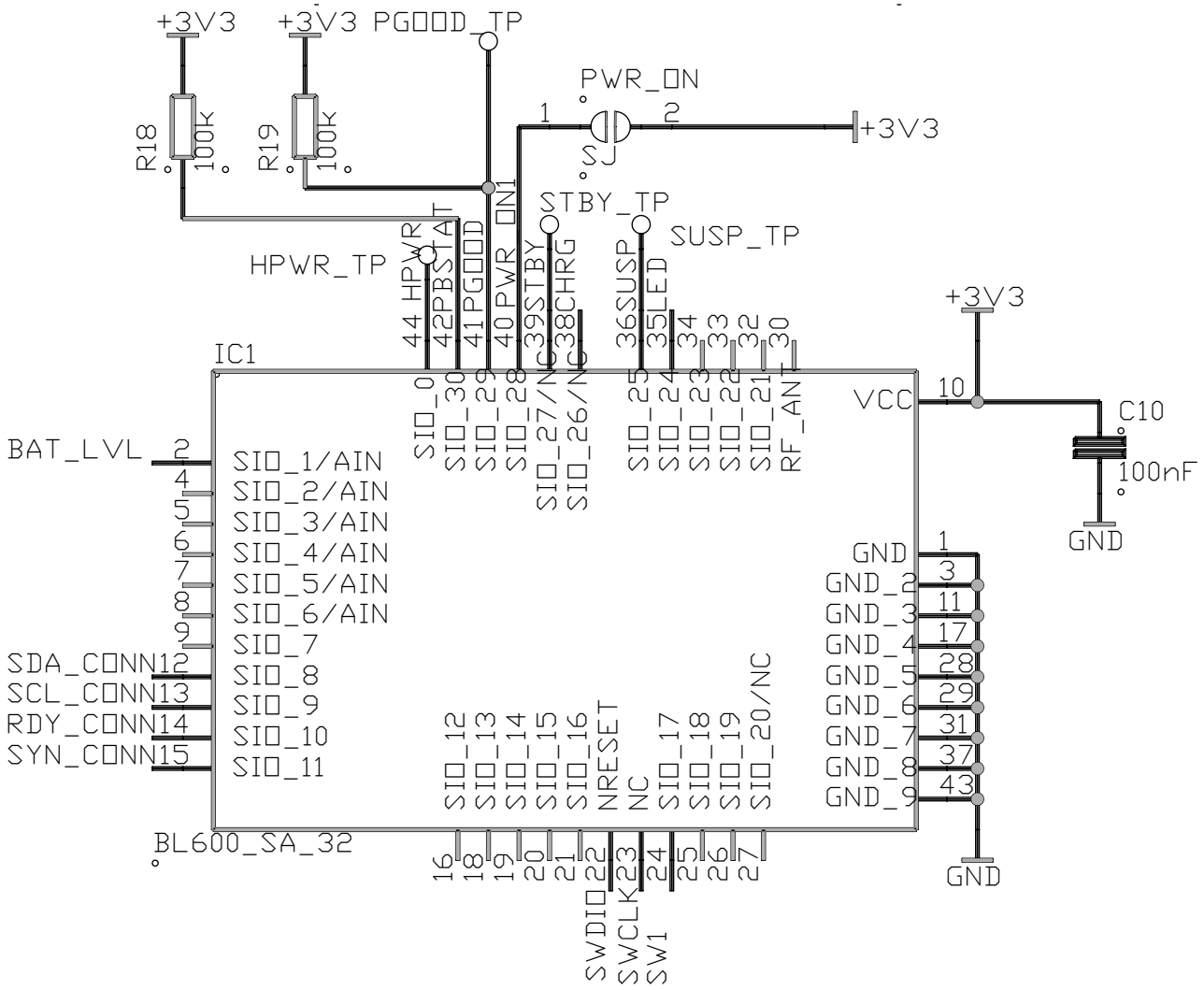
LISA B - Seadme elektroonikaskeem



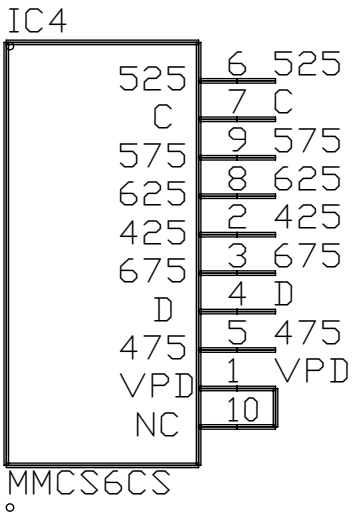
Teostas	Jan Bogdanov	Nimetus: Analoog-digitaalmuundur 1	
Kontrollis	Tanel Ainla		
Kinnitas	Tanel Ainla		
EMU TS - TN		Leht: 1/11	Tähis: TN 17/120463 B 01 01 S



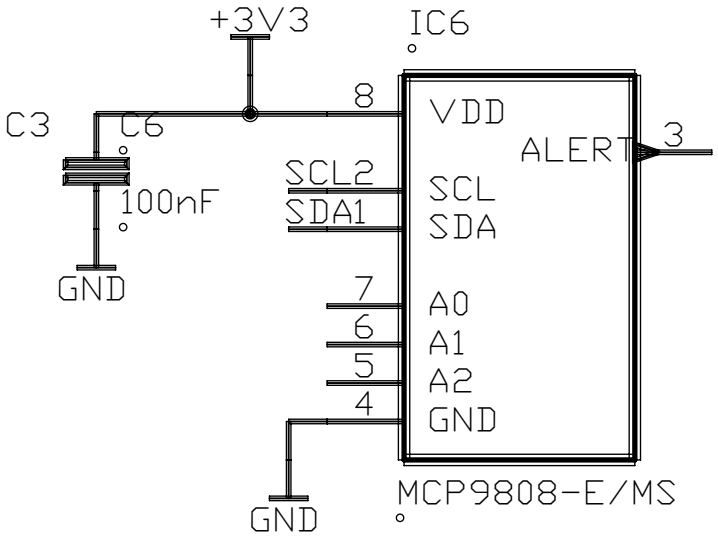
Teostas	Jan Bogdanov	Nimetus: Analoog-digitaalmuundur 2	
Kontrollis	Tanel Ainla		
Kinnitas	Tanel Ainla		
EMU TS - TN		Leht: 2/11	Tähis: TN 17/120463 B 01 02 S



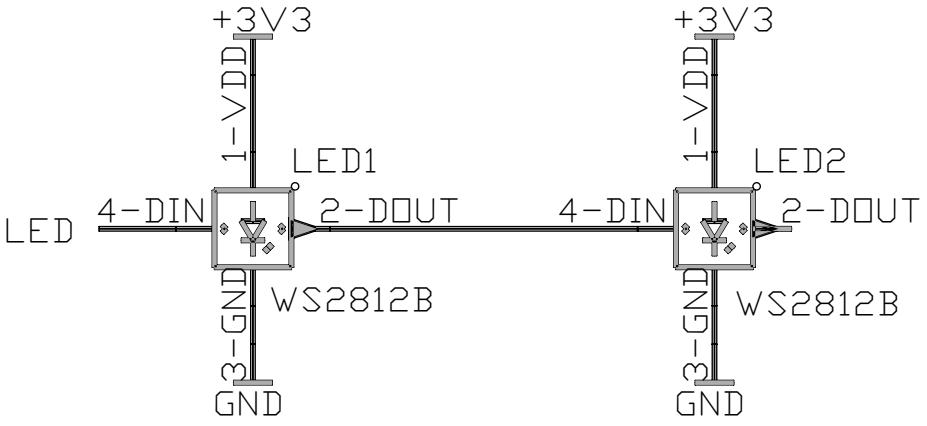
Teostas	Jan Bogdanov	Nimetus: Mikrokontroller/raadiosidemoodul	
Kontrollis	Tanel Ainla		
Kinnitas	Tanel Ainla		
EMU TS - TN		Leht: 3/11	Tähis: TN 17/120463 B 01 03 S



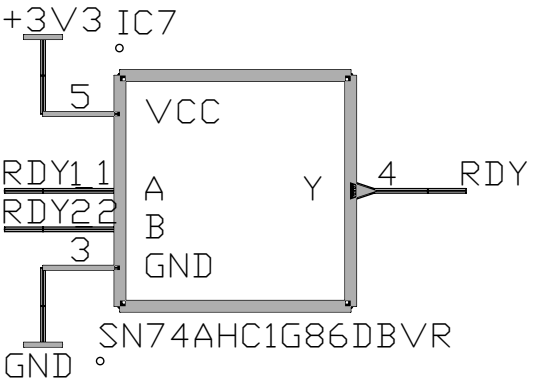
Teostas	Jan Bogdanov	Nimetus: Mitmikvärviandur	
Kontrollis	Tanel Ainla		
Kinnitas	Tanel Ainla		
EMU TS - TN		Leht: 4/11	Tähis: TN 17/120463 B 01 04 S

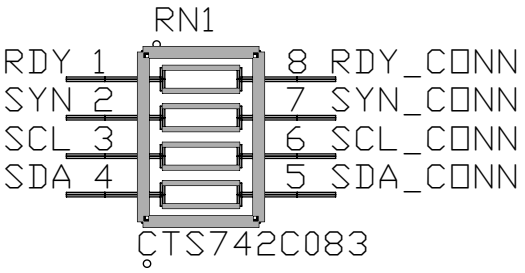


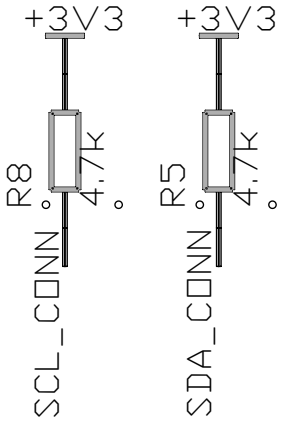
Teostas	Jan Bogdanov	Nimetus: Temperatuuriandur	
Kontrollis	Tanel Ainla		
Kinnitas	Tanel Ainla		
EMU TS - TN		Leht: 5/11	Tähis: TN 17/120463 B 01 05 S

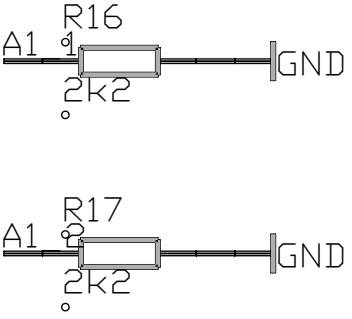


Teostas	Jan Bogdanov	Nimetus: Valgusdioodindikaatorid	
Kontrollis	Tanel Ainla		
Kinnitas	Tanel Ainla		
EMU TS - TN		Leht: 6/11	Tähis: TN 17/120463 B 01 06 S

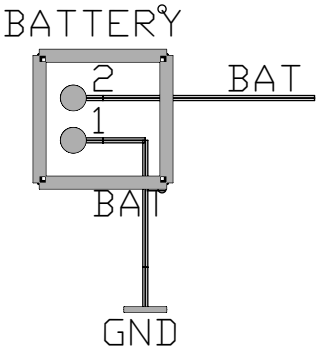


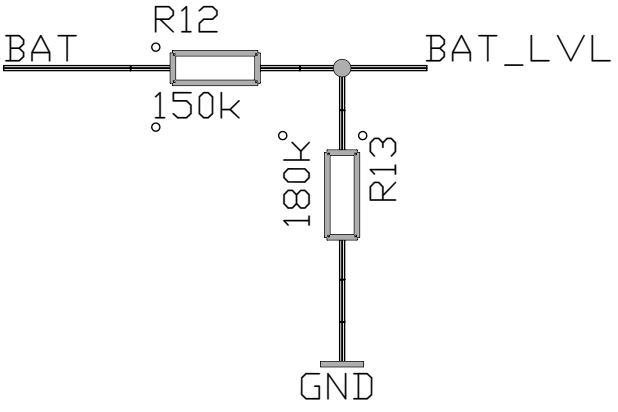




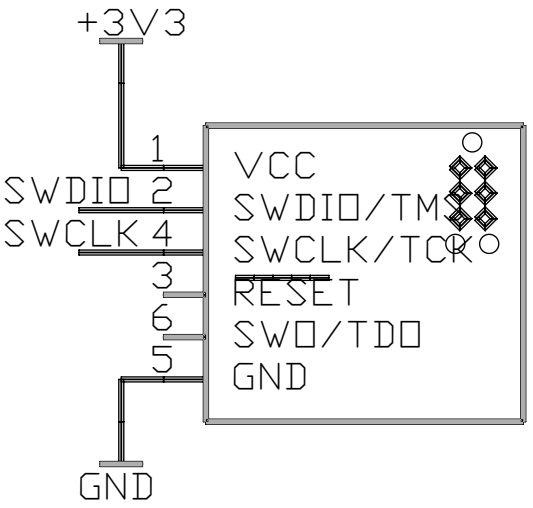


Teostas	Jan Bogdanov	Nimetus: I2C ja juhtsignaalide ahelad	
Kontrollis	Tanel Ainla		
Kinnitas	Tanel Ainla		
EMU TS - TN		Leht: 7/11	Tähis: TN 17/120463 B 01 07 S

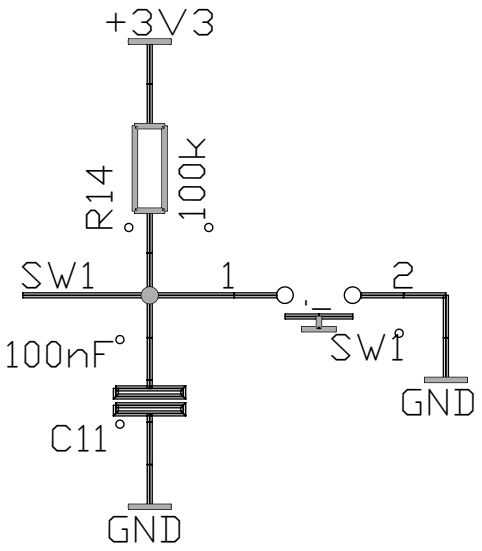




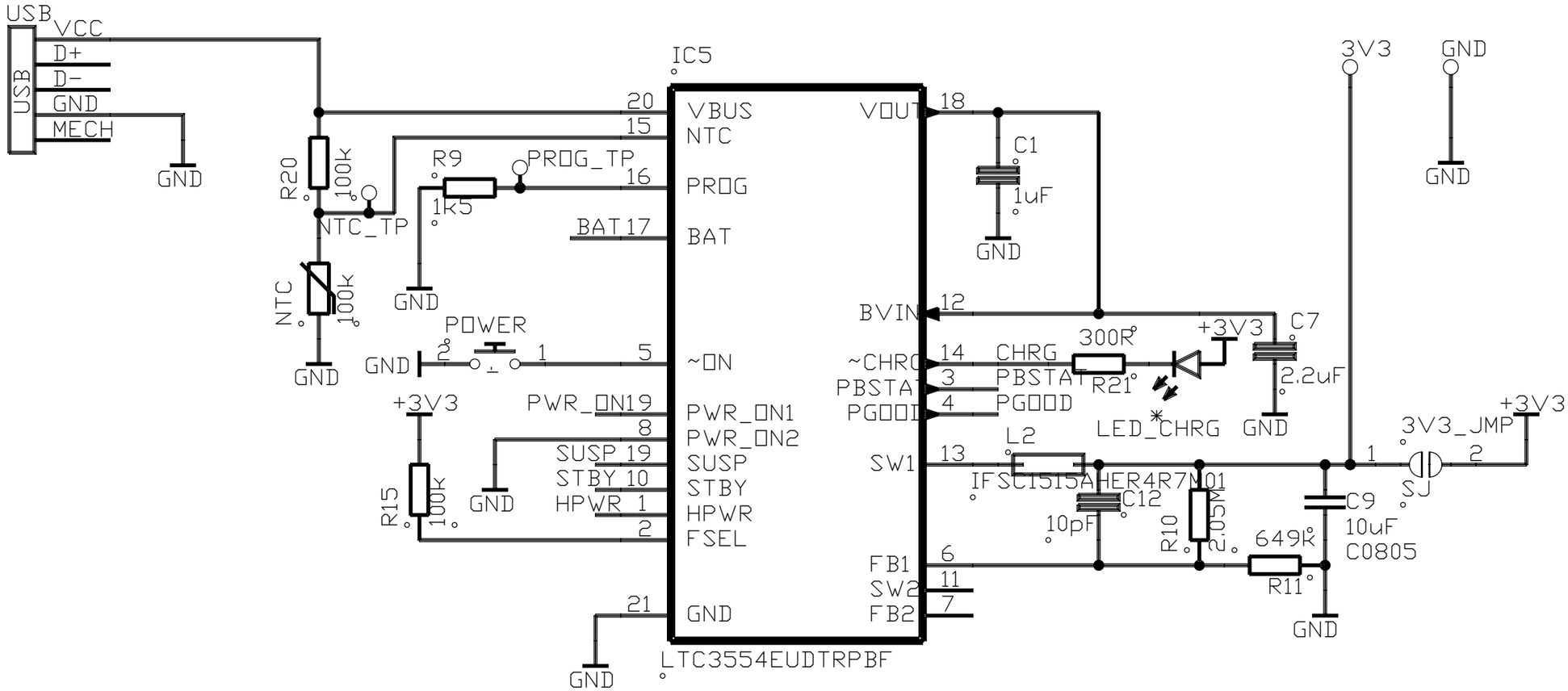
Teostas	Jan Bogdanov	Nimetus: Akupinge sobitamine ja juhtmepesa	
Kontrollis	Tanel Ainla		
Kinnitas	Tanel Ainla		
EMU TS - TN		Leht: 8/11	Tähis: TN 17/120463 B 01 08 S



Teostas	Jan Bogdanov	Nimetus: "Tag-connect" pesa ühendused	
Kontrollis	Tanel Ainla		
Kinnitas	Tanel Ainla		
EMU TS - TN		Leht: 9 /11	Tähis: TN 17/120463 B 01 09 S



Teostas	Jan Bogdanov	Nimetus: Surunupplüliti ahel	
Kontrollis	Tanel Ainla		
Kinnitas	Tanel Ainla		
EMU TS - TN		Leht: 10/11	Tähis: TN 17/120463 B 01 10 S



Teostas	Jan Bogdanov	Nimetus: Pingeregulaator ja akuhaldur	
Kontrollis	Tanel Ainla		
Kinnitas	Tanel Ainla		
EMU TS - TN		Leht: 11/11	Tähis: TN 17/120463 B 01 11 S

LISA C - Seadme püsivara

our_service.c

```
#include <stdint.h>
#include <string.h>
#include "nrf_gpio.h"
#include "our_service.h"
#include "ble_srv_common.h"
#include "app_error.h"

// Declaration of a function that will take care of some housekeeping of ble connections
    ↪ related to the service and characteristic
void ble_our_service_on_ble_evt(ble_os_t * p_our_service, ble_evt_t * p_ble_evt)
{
    // Implement switch case handling BLE events related to the service.
        switch (p_ble_evt->header.evt_id)
        {
            case BLE_GAP_EVT_CONNECTED:
                p_our_service->conn_handle = p_ble_evt->evt.gap_evt.conn_handle;
                break;
            case BLE_GAP_EVT_DISCONNECTED:
                p_our_service->conn_handle = BLE_CONN_HANDLE_INVALID;
                break;
            default:

                break;
        }
}

/* Function for adding the new characterstic to the service.
 *
 * Service structure.
 *
 */
static uint32_t our_char_add(ble_os_t * p_our_service)
{
    // Add a custom characteristic UUID
        uint32_t            err_code;
        ble_uuid_t          char_uuid;
        ble_uuid128_t       base_uuid = BLE_UUID_OUR_BASE_UUID;
        char_uuid.uuid      = BLE_UUID_OUR_CHARACTERISTC_UUID;
        err_code = sd_ble_uuid_vs_add(&base_uuid, &char_uuid.type);

        APP_ERROR_CHECK(err_code);

    // Add read/write properties to the characteristic
```

```
        ble_gatts_char_md_t char_md;
        memset(&char_md, 0, sizeof(char_md));
        char_md.char_props.read = 1;
        char_md.char_props.write = 1;

    // Configuring Client Characteristic Configuration Descriptor metadata and add to
    ↪ char_md structure
    ble_gatts_attr_md_t cccd_md;
    memset(&cccd_md, 0, sizeof(cccd_md));
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&cccd_md.read_perm);
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&cccd_md.write_perm);
    cccd_md.vloc                = BLE_GATTS_VLOC_STACK;
    char_md.p_cccd_md           = &cccd_md;
    char_md.char_props.notify   = 1;


    // Configure the attribute metadata
    ble_gatts_attr_md_t attr_md;
    memset(&attr_md, 0, sizeof(attr_md));
    attr_md.vloc        = BLE_GATTS_VLOC_STACK;

    // Set read/write security levels to our characteristic
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&attr_md.read_perm);
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&attr_md.write_perm);

    // Configure the characteristic value attribute
    ble_gatts_attr_t    attr_char_value;
    memset(&attr_char_value, 0, sizeof(attr_char_value));
    attr_char_value.p_uuid      = &char_uuid;
    attr_char_value.p_attr_md   = &attr_md;

    // Set characteristic length in number of bytes
    attr_char_value.max_len     = 17;
    ↪ ////////////////////////////////////-------------------16
    attr_char_value.init_len    = 17;
    ↪ ////////////////////////////////////-------------------16
    uint8_t value[4]            = {0x12,0x34,0x56,0x78};
    attr_char_value.p_value     = value;

    // Add our new characteristic to the service

    err_code = sd_ble_gatts_characteristic_add(p_our_service->service_handle,
                                       &char_md,
                                       &attr_char_value,
                                       &p_our_service->char_handles);
    APP_ERROR_CHECK(err_code);

    return NRF_SUCCESS;
}


/* Function for initiating our new service.
 *
 * Service structure.
```

```
 *
 */
void our_service_init(ble_os_t * p_our_service)
{
    uint32_t   err_code; // Variable to hold return codes from library and softdevice
    ↪ functions

    // Declare 16-bit service and 128-bit base UUIDs and add them to the BLE stack
    ble_uuid_t        service_uuid;
    ble_uuid128_t     base_uuid = BLE_UUID_OUR_BASE_UUID;
    service_uuid.uuid = BLE_UUID_OUR_SERVICE_UUID;
    err_code = sd_ble_uuid_vs_add(&base_uuid, &service_uuid.type);
    APP_ERROR_CHECK(err_code);

    // Set our service connection handle to default value. I.e. an invalid handle since
    ↪ we are not yet in a connection.
    p_our_service->conn_handle = BLE_CONN_HANDLE_INVALID;

    // Add our service
    err_code = sd_ble_gatts_service_add(BLE_GATTS_SRVC_TYPE_PRIMARY,
                                        &service_uuid,
                                        &p_our_service->service_handle);

    APP_ERROR_CHECK(err_code);

    // Call the function our_char_add() to add the new characteristic to the service.
    our_char_add(p_our_service);
}

// Function to be called when updating characteristic value

uint32_t characteristic_update(ble_os_t *p_our_service, uint8_t data[17])
{
        uint32_t err_code;
    // Update characteristic value
        if (p_our_service->conn_handle != BLE_CONN_HANDLE_INVALID)
        {
                uint16_t               len = 17;
                ble_gatts_hvx_params_t hvx_params;
                memset(&hvx_params, 0, sizeof(hvx_params));

                hvx_params.handle = p_our_service->char_handles.value_handle;
                hvx_params.type   = BLE_GATT_HVX_NOTIFICATION;
                hvx_params.offset = 0;
                hvx_params.p_len  = &len;
                hvx_params.p_data = data;

                err_code = sd_ble_gatts_hvx(p_our_service->conn_handle, &hvx_params);
    }
    else
    {
        err_code = NRF_ERROR_INVALID_STATE;
    }
```

```
    return err_code;
}
```

our_service.h

```
#ifndef OUR_SERVICE_H__
#define OUR_SERVICE_H__

#include <stdint.h>
#include "ble.h"
#include "ble_srv_common.h"

// Defining 16-bit service and 128-bit base UUIDs
#define BLE_UUID_OUR_BASE_UUID              {{0x23, 0xD1, 0x13, 0xEF, 0x5F, 0x78, 0x23,
    ↪ 0x15, 0xDE, 0xEF, 0x12, 0x12, 0x00, 0x00, 0x00, 0x00}} // 128-bit base UUID
#define BLE_UUID_OUR_SERVICE_UUID               0xF00D // Just a random, but
    ↪ recognizable value

// Defining 16-bit characteristic UUID
#define BLE_UUID_OUR_CHARACTERISTC_UUID         0xBEEF // Just a random, but
    ↪ recognizable value

typedef struct
{
    uint16_t                    conn_handle;    /**< Handle of the current connection (
    ↪ as provided by the BLE stack, is BLE_CONN_HANDLE_INVALID if not in a connection)
    ↪ .*/
    uint16_t                    service_handle; /**< Handle of Our Service (as provided
    ↪ by the BLE stack). */
    // Add handles for the characteristic attributes to struct
    ble_gatts_char_handles_t    char_handles;
}ble_os_t;

/**@brief Function for handling BLE Stack events related to the service and
    ↪ characteristic.
 *
 * @details Handles all events from the BLE stack of interest to Our Service.
 *
 * @param[in]   p_our_service       Our Service structure.
 * @param[in]   p_ble_evt  Event received from the BLE stack.
 */
void ble_our_service_on_ble_evt(ble_os_t * p_our_service, ble_evt_t * p_ble_evt);

/**@brief Function for initializing our new service.
 *
 * @param[in]   p_our_service       Pointer to Our Service structure.
 */
void our_service_init(ble_os_t * p_our_service);

/**@brief Function for updating and sending new characteristic values
 *
 * @details The application calls this function whenever our timer_timeout_handler
    ↪ triggers
 *
 * @param[in]   p_our_service                     Our Service structure.
 * @param[in]   characteristic_value     New characteristic value.
```

```
 */
uint32_t characteristic_update(ble_os_t *p_our_service, uint8_t data[1]);

#endif  /* _ OUR_SERVICE_H__ */
```

main.c

```
#include <stdint.h>
#include <string.h>
#include "nordic_common.h"
#include "nrf.h"
#include "app_error.h"
#include "ble.h"
#include "ble_hci.h"
#include "ble_srv_common.h"
#include "ble_advdata.h"
#include "ble_advertising.h"
#include "ble_conn_params.h"
#include "boards.h"
#include "softdevice_handler.h"
#include "app_timer.h"
#include "device_manager.h"
#include "pstorage.h"
#include "app_trace.h"
#include "bsp.h"
#include "bsp_btn_ble.h"
#include "our_service.h"
#include "nrf_delay.h"
#include "nrf_nvic.h"
#include "neopixel.h"


#define IS_SRVC_CHANGED_CHARACT_PRESENT  1                                          /**<
    ↪  Include or not the service_changed characteristic. if not enabled, the server's
    ↪ database cannot be changed for the lifetime of the device*/

#define CENTRAL_LINK_COUNT               0                                          /**<
    ↪ number of central links used by the application. When changing this number
    ↪ remember to adjust the RAM settings*/
#define PERIPHERAL_LINK_COUNT            1                                          /**<
    ↪ number of peripheral links used by the application. When changing this number
    ↪ remember to adjust the RAM settings*/

#define DEVICE_NAME                      "Nordic_Luminosity"
    ↪ /**< Name of device. Will be included in the advertising data. */
#define APP_ADV_INTERVAL                 300                                        /**<
    ↪  The advertising interval (in units of 0.625 ms. This value corresponds to 25 ms).
    ↪  */
#define APP_ADV_TIMEOUT_IN_SECONDS       180                                        /**<
    ↪  The advertising timeout in units of seconds. */

#define APP_TIMER_PRESCALER              0                                          /**<
    ↪  Value of the RTC1 PRESCALER register. */
#define APP_TIMER_OP_QUEUE_SIZE          4                                          /**<
    ↪  Size of timer operation queues. */
```

```
#define MIN_CONN_INTERVAL                MSEC_TO_UNITS(10, UNIT_1_25_MS)           /**<
    ↪ Minimum acceptable connection interval (0.1 seconds). */
#define MAX_CONN_INTERVAL                MSEC_TO_UNITS(200, UNIT_1_25_MS)           /**<
    ↪  Maximum acceptable connection interval (0.2 second). */
#define SLAVE_LATENCY                    0                                          /**<
    ↪  Slave latency. */
#define CONN_SUP_TIMEOUT                 MSEC_TO_UNITS(4000, UNIT_10_MS)            /**<
    ↪  Connection supervisory timeout (4 seconds). */

#define FIRST_CONN_PARAMS_UPDATE_DELAY   APP_TIMER_TICKS(5000, APP_TIMER_PRESCALER) /**<
    ↪  Time from initiating event (connect or start of notification) to first time
    ↪ sd_ble_gap_conn_param_update is called (5 seconds). */
#define NEXT_CONN_PARAMS_UPDATE_DELAY    APP_TIMER_TICKS(30000, APP_TIMER_PRESCALER)/**<
    ↪  Time between each call to sd_ble_gap_conn_param_update after the first call (30
    ↪ seconds). */
#define MAX_CONN_PARAMS_UPDATE_COUNT     3                                          /**<
    ↪  Number of attempts before giving up the connection parameter negotiation. */

#define SEC_PARAM_BOND                   1                                          /**<
    ↪  Perform bonding. */
#define SEC_PARAM_MITM                   0                                          /**<
    ↪  Man In The Middle protection not required. */
#define SEC_PARAM_IO_CAPABILITIES        BLE_GAP_IO_CAPS_NONE                       /**<
    ↪  No I/O capabilities. */
#define SEC_PARAM_OOB                    0                                          /**<
    ↪  Out Of Band data not available. */
#define SEC_PARAM_MIN_KEY_SIZE           7                                          /**<
    ↪  Minimum encryption key size. */
#define SEC_PARAM_MAX_KEY_SIZE           16                                         /**<
    ↪  Maximum encryption key size. */

#define DEAD_BEEF                        0xDEADBEEF                                 /**<
    ↪  Value used as error code on stack dump, can be used to identify stack location on
    ↪  stack unwind. */

#define LOW                     0
#define HIGH            1

#define SDA_PIN         8
#define SCL_PIN         9
#define RDY_PIN         12
#define TRG_PIN         11

#define SLAVE_1_WRITE   0b11101000
#define SLAVE_2_WRITE   0b11101010
#define SLAVE_1_READ    0b11101001
#define SLAVE_2_READ    0b11101011
#define OSR                             0x00
#define AGEN                    0x02
#define CREGL                   0x06
#define CREGH                   0x07
#define OPTREG                  0x08
#define BREAK                   0x09
#define EDGES                   0x0A
```

```
#define OSR_DEFAULT             0x42
#define AGEN_DEFAULT    0x20
#define CREGL_DEFAULT   0xB6
#define CREGH_DEFAULT   0x08
#define OPTREG_DEFAULT  0x00
#define BREAK_DEFAULT   0x20
#define EDGES_DEFAULT   0x01

#define I2C_SPEED       0

static dm_application_instance_t        m_app_handle;                               /**<
    ↪  Application identifier allocated by device manager */

static uint16_t                          m_conn_handle = BLE_CONN_HANDLE_INVALID;   /**<
    ↪  Handle of the current connection. */

uint8_t adc_bat = 0;
uint8_t adc_result = 0;

neopixel_strip_t m_strip;
uint8_t dig_pin_num = 24;
uint8_t leds_per_strip = 2;
uint8_t led_to_enable = 0;

bool led_status = 0;
uint8_t red = 0;
uint8_t green = 0;
uint8_t blue = 0;
//-----------------------------------I2C
    ↪ ----------------------------------------------------//
bool started = 0;
uint8_t package[16];
uint8_t output_data[17];
uint8_t n_bytes = 11;
bool read_SCL(void) {
    nrf_gpio_cfg_input(SCL_PIN, NRF_GPIO_PIN_PULLUP);
    return nrf_gpio_pin_read(SCL_PIN);
}
bool read_SDA(void) {
    nrf_gpio_cfg_input(SDA_PIN, NRF_GPIO_PIN_PULLUP);
    return nrf_gpio_pin_read(SDA_PIN);
}
bool read_RDY(void) {
    nrf_gpio_cfg_input(RDY_PIN, NRF_GPIO_PIN_PULLUP);
    return nrf_gpio_pin_read(RDY_PIN);
}
void set_SCL(void) {
    nrf_gpio_cfg_output(SCL_PIN);
    nrf_gpio_pin_set(SCL_PIN);
}
void clear_SCL(void) {
    nrf_gpio_cfg_output(SCL_PIN);
    nrf_gpio_pin_clear(SCL_PIN);
}
```

```
void set_SDA(void) {
    nrf_gpio_cfg_output(SDA_PIN);
    nrf_gpio_pin_set(SDA_PIN);
}
void clear_SDA(void) {
    nrf_gpio_cfg_output(SDA_PIN);
    nrf_gpio_pin_clear(SDA_PIN);
}
void i2c_delay (void) {
        volatile uint8_t v;
        for(uint16_t i = 0; i < I2C_SPEED; i++) {
                v += 0;
        }
}
void i2c_init(void) {
    nrf_gpio_cfg_output(RDY_PIN);
    nrf_gpio_pin_clear(RDY_PIN);
    nrf_gpio_cfg_output(TRG_PIN);
    nrf_gpio_pin_clear(TRG_PIN);
}
void i2c_start_cond( void )
{
    set_SDA();
    set_SCL();
    while( read_SCL() == 0 )
    {
        // Clock stretching
    }
  i2c_delay();
  clear_SDA();
  i2c_delay();
  clear_SCL();
}
void i2c_stop_cond( void )
{
  clear_SDA();
  set_SCL();
  while(read_SCL() == 0)
  {
          // Clock stretching
  }
  i2c_delay();
  set_SDA();
  i2c_delay();
  if(read_SDA() == 0)
  {
  }
  i2c_delay();
  started = 0;
}
void i2c_write_bit(bool bit) {
        if (bit == 1) {
                set_SDA();
        }
```

```
        else {
                clear_SDA();
        }
        i2c_delay();
        set_SCL();
        i2c_delay();
        while(read_SCL() == 0)
        {
                //Clock stretching
        }
        i2c_delay();
        clear_SCL();
        i2c_delay();
}
bool i2c_read_bit(void) {
        bool bit = 0;
        i2c_delay();
        nrf_gpio_cfg_input(SDA_PIN, NRF_GPIO_PIN_PULLUP);
        i2c_delay();
        set_SCL();
        while( read_SCL() == 0 )
        {
                // Clock stretching
        }
        i2c_delay();
        clear_SCL();
        bit = nrf_gpio_pin_read(SDA_PIN);
        return bit;
}
bool i2c_write_byte(bool send_start, bool send_stop, uint8_t byte) {
        bool nack = 0;
        if(send_start == 1)  {
            i2c_start_cond();
          }
        for(uint8_t i = 0; i < 8; i++) {
                i2c_write_bit((byte & 0x80) != 0);
                byte <<= 1;
        }
        nack = i2c_read_bit();
        i2c_delay();
        if (send_stop == 1) {
                i2c_stop_cond();
        }
        return nack;
}
uint8_t i2c_read_byte(bool nack, bool send_stop) {
        uint8_t byte = 0;

        for(uint8_t i = 0; i < 8; i++) {
                byte = ( byte << 1 ) | i2c_read_bit();
        }
          if(send_stop == 1)
          {
            i2c_stop_cond();
```

```
          }
          else {
                i2c_write_bit(nack);
          }
        return byte;
}
void init_SLAVE_1(void) {
        //-----------------------//
        i2c_write_byte(1, 0, 0xE8); //SLAVE_1_WRITE
        i2c_write_byte(0, 0, 0x00);     //OSR
        i2c_write_byte(0, 1, 0x02);     //CONF_MODE
        //-----------------------//
        i2c_write_byte(1, 0, 0xEA);     //SLAVE_2_WRITE
        i2c_write_byte(0, 0, 0x00); //OSR
        i2c_write_byte(0, 1, 0x02);     //CONF_MODE
        //-----------------------//
        i2c_write_byte(1, 0, 0xE8);     //SLAVE_1_WRITE
        i2c_write_byte(0, 0, 0x00);     //OSR
        i2c_write_byte(0, 1, 0x00);
        //-----------------------//
        i2c_write_byte(1, 0, 0xEA); //SLAVE_2_WRITE
        i2c_write_byte(0, 0, 0x00); //OSR
        i2c_write_byte(0, 1, 0x00);
        //-----------------------//
        i2c_write_byte(1, 0, 0xE8); //SLAVE_1_WRITE
        i2c_write_byte(0, 0, 0x06); //CREGL
        i2c_write_byte(0, 1, 0x80);     //CREGL_DEFAULT
        //-----------------------//
        i2c_write_byte(1, 0, 0xEA);     //SLAVE_2_WRITE
        i2c_write_byte(0, 0, 0x06);     //CREGL
        i2c_write_byte(0, 1, 0x80);     //CREGL_DEFAULT /B6
        //-----------------------//
        i2c_write_byte(1, 0, 0xE8);     //SLAVE_1_WRITE
        i2c_write_byte(0, 0, 0x07); //CREGH
        i2c_write_byte(0, 1, 0x08); //CREGH_DEFAULT
        //-----------------------//
        i2c_write_byte(1, 0, 0xEA); //SLAVE_2_WRITE
        i2c_write_byte(0, 0, 0x07); //CREGH
        i2c_write_byte(0, 1, 0x08);     //CREGH_DEFAULT
        //-----------------------//
        i2c_write_byte(1, 0, 0xE8); //SLAVE_1_WRITE
        i2c_write_byte(0, 0, 0x08); //OPTREG
        i2c_write_byte(0, 1, 0x00); //OPTREG_DEFAULT
        //-----------------------//
        i2c_write_byte(1, 0, 0xEA); //SLAVE_2_WRITE
        i2c_write_byte(0, 0, 0x08); //OPTREG
        i2c_write_byte(0, 1, 0x00); //OPTREG_DEFAULT
        //-----------------------//
        i2c_write_byte(1, 0, 0xE8); //SLAVE_1_WRITE
        i2c_write_byte(0, 1, 0x00); //OSR
        //-----------------------//
        i2c_write_byte(1, 0, 0xEA); //SLAVE_2_WRITE
        i2c_write_byte(0, 1, 0x00); //OSR
        //-----------------------//
```

```
        i2c_write_byte(1, 0, 0xE9); //SLAVE_1_READ
        n_bytes = 11;
        for (uint8_t i = 0; i < n_bytes; i++) {
                if (i == (n_bytes - 1)) {
                        package[i] = i2c_read_byte(1, 1);
                }
                else {
                        package[i] = i2c_read_byte(0, 0);
                }
        }
        //-----------------------//
        i2c_write_byte(1, 0, 0xEB); //SLAVE_2_READ
        n_bytes = 11;
        for (uint8_t i = 0; i < n_bytes; i++) {
                if (i == (n_bytes - 1)) {
                        package[i] = i2c_read_byte(1, 1);
                }
                else {
                        package[i] = i2c_read_byte(0, 0);
                }
        }
        //-----------------------//
        i2c_write_byte(1, 0, 0xE8);
        i2c_write_byte(0, 1, 0x07);
        //-----------------------//
        i2c_write_byte(1, 0, 0xE9); //SLAVE_1_READ
        n_bytes = 3;
        for (uint8_t i = 0; i < n_bytes; i++) {
                if (i == (n_bytes - 1)) {
                        package[i] = i2c_read_byte(1, 1);
                }
                else {
                        package[i] = i2c_read_byte(0, 0);
                }
        }
        //-----------------------//
        i2c_write_byte(1, 0, 0xE8); //
        i2c_write_byte(0, 1, 0x06); //
        //-----------------------//
        i2c_write_byte(1, 0, 0xE9); //SLAVE_1_READ
        n_bytes = 3;
        for (uint8_t i = 0; i < n_bytes; i++) {
                if (i == (n_bytes - 1)) {
                        package[i] = i2c_read_byte(1, 1);
                }
                else {
                        package[i] = i2c_read_byte(0, 0);
                }
        }
        //-----------------------//
        i2c_write_byte(1, 0, 0xE8); //
        i2c_write_byte(0, 1, 0x07); //
        //-----------------------//
        i2c_write_byte(1, 0, 0xE9); //SLAVE_1_READ
```

```
        n_bytes = 3;
        for (uint8_t i = 0; i < n_bytes; i++) {
                if (i == (n_bytes - 1)) {
                        package[i] = i2c_read_byte(1, 1);
                }
                else {
                        package[i] = i2c_read_byte(0, 0);
                }
        }
        //-----------------------//
        i2c_write_byte(1, 0, 0xE8); //
        i2c_write_byte(0, 1, 0x08); //
        //-----------------------//
        i2c_write_byte(1, 0, 0xE9); //SLAVE_1_READ
        n_bytes = 3;
        for (uint8_t i = 0; i < n_bytes; i++) {
                if (i == (n_bytes - 1)) {
                        package[i] = i2c_read_byte(1, 1);
                }
                else {
                        package[i] = i2c_read_byte(0, 0);
                }
        }
        //-----------------------//
        i2c_write_byte(1, 0, 0xE8); //
        i2c_write_byte(0, 1, 0x03); //
        //-----------------------//
        i2c_write_byte(1, 0, 0xE9); //SLAVE_1_READ
        n_bytes = 3;
        for (uint8_t i = 0; i < n_bytes; i++) {
                if (i == (n_bytes - 1)) {
                        package[i] = i2c_read_byte(1, 1);
                }
                else {
                        package[i] = i2c_read_byte(0, 0);
                }
        }
}
void measure(void) {
        //-----------------------//
        i2c_write_byte(1, 0, 0xE8); //
        i2c_write_byte(0, 0, 0x00); //
        i2c_write_byte(0, 1, 0x83); //
        //-----------------------//
        i2c_write_byte(1, 0, 0xEA); //
        i2c_write_byte(0, 0, 0x00); //
        i2c_write_byte(0, 1, 0x83); //
        //-----------------------//
        i2c_write_byte(1, 0, 0xE8); //
        i2c_write_byte(0, 1, 0x00); //
        //-----------------------//
        i2c_write_byte(1, 0, 0xEA); //
        i2c_write_byte(0, 1, 0x00); //
        //-----------------------//
```

```
}
void read(void) {
        i2c_write_byte(1, 0, 0xE9); //SLAVE_1_READ
        n_bytes = 8;
        for (uint8_t i = 0; i < n_bytes; i++) {
                if (i == (n_bytes - 1)) {
                        package[i] = i2c_read_byte(1, 1);
                }
                else {
                        package[i] = i2c_read_byte(0, 0);
                }
        }
        i2c_write_byte(1, 0, 0xEB); //SLAVE_2_READ
        n_bytes = 8;
        for (uint8_t i = n_bytes; i < (n_bytes + n_bytes); i++) {
                if (i == (n_bytes + n_bytes - 1)) {
                        package[i] = i2c_read_byte(1, 1);
                }
                else {
                        package[i] = i2c_read_byte(0, 0);
                }
        }
}

void convert_data() {
        output_data[0] = package[9];                    //P425
        output_data[1] = package[8];
        output_data[2] = package[15]; //P475
        output_data[3] = package[14];
        output_data[4] = package[1];            //P525
        output_data[5] = package[0];
        output_data[6] = package[7];            //P575
        output_data[7] = package[6];
        output_data[8] = package[5];            //P625
        output_data[9] = package[4];
        output_data[10] = package[11];          //P675
        output_data[11] = package[10];
        output_data[12] = package[3];           //CENTRE
        output_data[13] = package[2];
        output_data[14] = package[13];          //DARK
        output_data[15] = package[12];
}

/*
void convert_data() {                   // Fot testing
        output_data[0] = 0x01;
        output_data[1] = 0x02;
        output_data[2] = 0x03;
        output_data[3] = 0x04;
        output_data[4] = 0x05;
        output_data[5] = 0x06;
        output_data[6] = 0x07;
        output_data[7] = 0x08;
        output_data[8] = 0x09;
```

```
        output_data[9] = 0x10;
        output_data[10] = 0x11;
        output_data[11] = 0x12;
        output_data[12] = 0x13;
        output_data[13] = 0x14;
        output_data[14] = 0x15;
        output_data[15] = 0x16;
}
*/
//-----------------------------------I2C
    ↪ ----------------------------------------------------//

//-----------------------------------------LED
    ↪ ---------------------------------------------//

void set_led_colour(uint8_t r, uint8_t g, uint8_t b) {
        red = r;
        green = g;
        blue = b;
        led_status = 1;
}


//-----------------------------------------LED
    ↪ ---------------------------------------------//

//-----------------------------------------ADC
    ↪ ---------------------------------------------//
void adc_1()
{
        // interrupt ADC
        NRF_ADC->INTENSET = (ADC_INTENSET_END_Disabled << ADC_INTENSET_END_Pos);
    ↪                                       /*!< Interrupt enabled. */

        // config ADC
        NRF_ADC->CONFIG = (ADC_CONFIG_EXTREFSEL_None << ADC_CONFIG_EXTREFSEL_Pos) /*
    ↪ Bits 17..16 : ADC external reference pin selection. */
                                                                                | (
    ↪ ADC_CONFIG_PSEL_AnalogInput2 << ADC_CONFIG_PSEL_Pos)
    ↪         /*!< Use analog input 1 as analog input. */
                                                                                | (
    ↪ ADC_CONFIG_REFSEL_VBG << ADC_CONFIG_REFSEL_Pos)
    ↪                         /*!< Use internal 1.2V bandgap voltage as reference for
    ↪ conversion. */
                                                                                | (
    ↪ ADC_CONFIG_INPSEL_AnalogInputOneThirdPrescaling << ADC_CONFIG_INPSEL_Pos) /*!<
    ↪ Analog input specified by PSEL with no prescaling used as input for the conversion
    ↪ . */
                                                                                | (
    ↪ ADC_CONFIG_RES_8bit << ADC_CONFIG_RES_Pos);
    ↪                                   /*!< 10bit ADC resolution. */

        // enable ADC
```

```
        NRF_ADC->ENABLE = ADC_ENABLE_ENABLE_Enabled;
    ↪
    ↪                           /* Bit 0 : ADC enable. */

        // start ADC conversion
        NRF_ADC->TASKS_START = 1;

        // wait for conversion to end
        while (!NRF_ADC->EVENTS_END)
        {}
        NRF_ADC->EVENTS_END     = 0;

        //Save your ADC result
        adc_result = NRF_ADC->RESULT;


        //Use the STOP task to save current. Workaround for PAN_028 rev1.1 anomaly 1.
  NRF_ADC->TASKS_STOP = 1;
}

//----------------------------------------ADC
    ↪ ----------------------------------------//
// Declare a service structure for the application
ble_os_t m_our_service;

void send_bulk_data();

// Declare an app_timer id variable and define timer interval and define a timer
    ↪ interval
APP_TIMER_DEF(m_our_char_timer_id);
#define OUR_CHAR_TIMER_INTERVAL     APP_TIMER_TICKS(500, APP_TIMER_PRESCALER) // 1000 ms
    ↪  intervals

APP_TIMER_DEF(m_our_char_timer_id_2);

static ble_uuid_t       m_adv_uuids[] = {{BLE_UUID_OUR_SERVICE_UUID,
    ↪ BLE_UUID_TYPE_VENDOR_BEGIN}};

/**@brief Callback function for asserts in the SoftDevice.
 *
 * @details This function will be called in case of an assert in the SoftDevice.
 *
 * @warning This handler is an example only and does not fit a final product. You need
    ↪ to analyze
 *          how your product is supposed to react in case of Assert.
 * @warning On assert from the SoftDevice, the system can only recover on reset.
 *
 * @param[in] line_num   Line number of the failing ASSERT call.
 * @param[in] file_name  File name of the failing ASSERT call.
 */
void assert_nrf_(uint16_t line_num, const uint8_t * p_file_name)
{
    app_error_handler(DEAD_BEEF, line_num, p_file_name);
}
```

```
// This is a timer event handler
static void timer_timeout_handler(void * p_context)
{
        send_bulk_data();
}

static void timer_timeout_handler_2(void * p_context)
{
        app_timer_start(m_our_char_timer_id, OUR_CHAR_TIMER_INTERVAL, NULL);

}

void send_bulk_data(void) {

    // Update characteristic value.
        uint32_t        err_code = 0;
    measure();
    read();
    convert_data();
    adc_1();
    output_data[16] = adc_result;

                characteristic_update(&m_our_service, output_data);
                if (//(err_code != NRF_SUCCESS) &&
                                        (err_code != NRF_ERROR_INVALID_STATE) &&
                                        (err_code != BLE_ERROR_NO_TX_PACKETS)// &&
                                        //(err_code != BLE_ERROR_GATTS_SYS_ATTR_MISSING)
                                        ){}

}

/**@brief Function for the Timer initialization.
 *
 * @details Initializes the timer module. This creates and starts application timers.
 */
static void timers_init(void)
{

    // Initialize timer module.
    APP_TIMER_INIT(APP_TIMER_PRESCALER, APP_TIMER_OP_QUEUE_SIZE, false);

    // Initiate our timer
    app_timer_create(&m_our_char_timer_id, APP_TIMER_MODE_REPEATED,
    ↪ timer_timeout_handler);
    app_timer_create(&m_our_char_timer_id_2, APP_TIMER_MODE_REPEATED,
    ↪ timer_timeout_handler_2);
}

/**@brief Function for the GAP initialization.
 *
 * @details This function sets up all the necessary GAP (Generic Access Profile)
    ↪ parameters of the
```

```
 *          device including the device name, appearance, and the preferred connection
    ↪ parameters.
 */
static void gap_params_init(void)
{
    uint32_t                err_code;
    ble_gap_conn_params_t   gap_conn_params;
    ble_gap_conn_sec_mode_t sec_mode;

    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&sec_mode);

    err_code = sd_ble_gap_device_name_set(&sec_mode,
                                          (const uint8_t *)DEVICE_NAME,
                                          strlen(DEVICE_NAME));
    APP_ERROR_CHECK(err_code);

    memset(&gap_conn_params, 0, sizeof(gap_conn_params));

    gap_conn_params.min_conn_interval = MIN_CONN_INTERVAL;
    gap_conn_params.max_conn_interval = MAX_CONN_INTERVAL;
    gap_conn_params.slave_latency     = SLAVE_LATENCY;
    gap_conn_params.conn_sup_timeout  = CONN_SUP_TIMEOUT;

    err_code = sd_ble_gap_ppcp_set(&gap_conn_params);
    APP_ERROR_CHECK(err_code);
}

/**@brief Function for initializing services that will be used by the application.
 */
static void services_init(void)
{
    // Add code to initialize the services used by the application.
    our_service_init(&m_our_service);
}

/**@brief Function for handling the Connection Parameters Module.
 *
 * @details This function will be called for all events in the Connection Parameters
    ↪ Module which
 *          are passed to the application.
 *          @note All this function does is to disconnect. This could have been done by
    ↪ simply
 *                setting the disconnect_on_fail config parameter, but instead we use
    ↪ the event
 *                handler mechanism to demonstrate its use.
 *
 * @param[in] p_evt  Event received from the Connection Parameters Module.
 */
static void on_conn_params_evt(ble_conn_params_evt_t * p_evt)
{
    uint32_t err_code;

    if (p_evt->evt_type == BLE_CONN_PARAMS_EVT_FAILED)
    {
```

```
        err_code = sd_ble_gap_disconnect(m_conn_handle,
    ↪ BLE_HCI_CONN_INTERVAL_UNACCEPTABLE);
        APP_ERROR_CHECK(err_code);
    }
}

/**@brief Function for handling a Connection Parameters error.
 *
 * @param[in] nrf_error  Error code containing information about what went wrong.
 */
static void conn_params_error_handler(uint32_t nrf_error)
{
    APP_ERROR_HANDLER(nrf_error);
}

/**@brief Function for initializing the Connection Parameters module.
 */
static void conn_params_init(void)
{
    uint32_t               err_code;
    ble_conn_params_init_t cp_init;

    memset(&cp_init, 0, sizeof(cp_init));

    cp_init.p_conn_params                  = NULL;
    cp_init.first_conn_params_update_delay = FIRST_CONN_PARAMS_UPDATE_DELAY;
    cp_init.next_conn_params_update_delay  = NEXT_CONN_PARAMS_UPDATE_DELAY;
    cp_init.max_conn_params_update_count   = MAX_CONN_PARAMS_UPDATE_COUNT;
    cp_init.start_on_notify_cccd_handle    = BLE_GATT_HANDLE_INVALID;
    cp_init.disconnect_on_fail             = false;
    cp_init.evt_handler                    = on_conn_params_evt;
    cp_init.error_handler                  = conn_params_error_handler;

    err_code = ble_conn_params_init(&cp_init);
    APP_ERROR_CHECK(err_code);
}

/**@brief Function for starting timers.
*/
static void application_timers_start(void)
{
    // Start timer
        app_timer_start(m_our_char_timer_id_2, OUR_CHAR_TIMER_INTERVAL, NULL);
}

/**@brief Function for putting the chip into sleep mode.
 *
 * @note This function will not return.
 */
static void sleep_mode_enter(void)
{
    uint32_t err_code = bsp_indication_set(BSP_INDICATE_IDLE);
    APP_ERROR_CHECK(err_code);
```

```
    // Prepare wakeup buttons.
    err_code = bsp_btn_ble_sleep_mode_prepare();
    APP_ERROR_CHECK(err_code);

    // Go to system-off mode (this function will not return; wakeup will cause a reset).
    err_code = sd_power_system_off();
    APP_ERROR_CHECK(err_code);
}

/**@brief Function for handling advertising events.
 *
 * @details This function will be called for advertising events which are passed to the
    ↪ application.
 *
 * @param[in] ble_adv_evt  Advertising event.
 */
static void on_adv_evt(ble_adv_evt_t ble_adv_evt)
{
    uint32_t err_code;

    switch (ble_adv_evt)
    {
        case BLE_ADV_EVT_FAST:
            err_code = bsp_indication_set(BSP_INDICATE_ADVERTISING);
            APP_ERROR_CHECK(err_code);
            break;
        case BLE_ADV_EVT_IDLE:
                led_to_enable = 0;
                set_led_colour(0, 50, 0);
            sleep_mode_enter();
            break;
        default:
            break;
    }
}

/**@brief Function for handling the Application's BLE Stack events.
 *
 * @param[in] p_ble_evt  Bluetooth stack event.
 */
static void on_ble_evt(ble_evt_t * p_ble_evt)
{
    uint32_t err_code;

    switch (p_ble_evt->header.evt_id)
            {
        case BLE_GAP_EVT_CONNECTED:
            err_code = bsp_indication_set(BSP_INDICATE_CONNECTED);
            APP_ERROR_CHECK(err_code);
            m_conn_handle = p_ble_evt->evt.gap_evt.conn_handle;
            led_to_enable = 0;
            set_led_colour(50, 3, 50);
            break;
```

```
        case BLE_GAP_EVT_DISCONNECTED:
            m_conn_handle = BLE_CONN_HANDLE_INVALID;
            led_to_enable = 0;
            set_led_colour(0, 0, 50);
            break;

        /*case BLE_EVT_TX_COMPLETE:
                if (count < 1000) {
                send_bulk_data();
                break;
                }
                else {
                        //count_2++;
                        count = 0;
                        app_timer_stop(m_our_char_timer_id);
                        break;
                }*/



        default:
            // No implementation needed.
            break;
    }
}

/**@brief Function for dispatching a BLE stack event to all modules with a BLE stack
    ↪ event handler.
 *
 * @details This function is called from the BLE Stack event interrupt handler after a
    ↪ BLE stack
 *          event has been received.
 *
 * @param[in] p_ble_evt  Bluetooth stack event.
 */
static void ble_evt_dispatch(ble_evt_t * p_ble_evt)
{
    dm_ble_evt_handler(p_ble_evt);
    ble_conn_params_on_ble_evt(p_ble_evt);
    bsp_btn_ble_on_ble_evt(p_ble_evt);
    on_ble_evt(p_ble_evt);
    ble_advertising_on_ble_evt(p_ble_evt);
    // Call ble_our_service_on_ble_evt() to do housekeeping of ble connections related
    ↪ to the service and characteristic
    ble_our_service_on_ble_evt(&m_our_service, p_ble_evt);
}

/**@brief Function for dispatching a system event to interested modules.
 *
 * @details This function is called from the System event interrupt handler after a
    ↪ system
 *          event has been received.
 *
 * @param[in] sys_evt  System stack event.
```

```
 */
static void sys_evt_dispatch(uint32_t sys_evt)
{
    pstorage_sys_event_handler(sys_evt);
    ble_advertising_on_sys_evt(sys_evt);
}

/**@brief Function for initializing the BLE stack.
 *
 * @details Initializes the SoftDevice and the BLE event interrupt.
 */
static void ble_stack_init(void)
{
    uint32_t err_code;

    nrf_clock_lf_cfg_t clock_lf_cfg =
    {
        .source = NRF_CLOCK_LF_SRC_RC,
        .rc_ctiv = 16, // Interval in 0.25 s, 16 * 0.25 = 4 sec
        .rc_temp_ctiv = 2, // Check temperature every .rc_ctiv, but calibrate every .
    ↪ rc_temp_ctiv
        .xtal_accuracy = NRF_CLOCK_LF_XTAL_ACCURACY_250_PPM,
    };
    //nrf_clock_lf_cfg_t clock_lf_cfg = NRF_CLOCK_LF_SRC_SYNTH;
    // Initialize the SoftDevice handler module.
    SOFTDEVICE_HANDLER_INIT(&clock_lf_cfg, NULL);

    ble_enable_params_t ble_enable_params;
    err_code = softdevice_enable_get_default_config(CENTRAL_LINK_COUNT,
                                                    PERIPHERAL_LINK_COUNT,
                                                    &ble_enable_params);
    APP_ERROR_CHECK(err_code);

    //Check the ram settings against the used number of links
    CHECK_RAM_START_ADDR(CENTRAL_LINK_COUNT,PERIPHERAL_LINK_COUNT);

    // Enable BLE stack.
    ble_enable_params.gatts_enable_params.service_changed =
    ↪ IS_SRVC_CHANGED_CHARACT_PRESENT;
    err_code = softdevice_enable(&ble_enable_params);
    APP_ERROR_CHECK(err_code);

    // Register with the SoftDevice handler module for BLE events.
    err_code = softdevice_ble_evt_handler_set(ble_evt_dispatch);
    APP_ERROR_CHECK(err_code);

    // Register with the SoftDevice handler module for BLE events.
    err_code = softdevice_sys_evt_handler_set(sys_evt_dispatch);
    APP_ERROR_CHECK(err_code);
}

/**@brief Function for handling events from the BSP module.
 *
 * @param[in]   event   Event generated by button press.
```

```
 */
void bsp_event_handler(bsp_event_t event)
{
    uint32_t err_code;
    switch (event)
    {
        case BSP_EVENT_SLEEP:
                led_to_enable = 0;
                set_led_colour(50, 50, 0);
            sleep_mode_enter();
            break;

        case BSP_EVENT_DISCONNECT:
            err_code = sd_ble_gap_disconnect(m_conn_handle,
    ↪ BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
            if (err_code != NRF_ERROR_INVALID_STATE)
            {
                APP_ERROR_CHECK(err_code);
            }
            break;

        case BSP_EVENT_WHITELIST_OFF:
            err_code = ble_advertising_restart_without_whitelist();
            if (err_code != NRF_ERROR_INVALID_STATE)
            {
                APP_ERROR_CHECK(err_code);
            }
            break;

        default:
            break;
    }
}

/**@brief Function for handling the Device Manager events.
 *
 * @param[in] p_evt  Data associated to the device manager event.
 */
static uint32_t device_manager_evt_handler(dm_handle_t const * p_handle,
                                           dm_event_t const  * p_event,
                                           ret_code_t        event_result)
{
    APP_ERROR_CHECK(event_result);

#ifdef BLE_DFU_APP_SUPPORT
    if (p_event->event_id == DM_EVT_LINK_SECURED)
    {
        app_context_load(p_handle);
    }
#endif // BLE_DFU_APP_SUPPORT

    return NRF_SUCCESS;
}
```

```
/**@brief Function for the Device Manager initialization.
 *
 * @param[in] erase_bonds  Indicates whether bonding information should be cleared from
 *                         persistent storage during initialization of the Device
    ↪ Manager.
 */
static void device_manager_init(bool erase_bonds)
{
    uint32_t               err_code;
    dm_init_param_t        init_param = {.clear_persistent_data = erase_bonds};
    dm_application_param_t register_param;

    // Initialize persistent storage module.
    err_code = pstorage_init();
    APP_ERROR_CHECK(err_code);

    err_code = dm_init(&init_param);
    APP_ERROR_CHECK(err_code);

    memset(&register_param.sec_param, 0, sizeof(ble_gap_sec_params_t));

    register_param.sec_param.bond         = SEC_PARAM_BOND;
    register_param.sec_param.mitm         = SEC_PARAM_MITM;
    register_param.sec_param.io_caps      = SEC_PARAM_IO_CAPABILITIES;
    register_param.sec_param.oob          = SEC_PARAM_OOB;
    register_param.sec_param.min_key_size = SEC_PARAM_MIN_KEY_SIZE;
    register_param.sec_param.max_key_size = SEC_PARAM_MAX_KEY_SIZE;
    register_param.evt_handler            = device_manager_evt_handler;
    register_param.service_type           = DM_PROTOCOL_CNTXT_GATT_SRVR_ID;

    err_code = dm_register(&m_app_handle, &register_param);
    APP_ERROR_CHECK(err_code);
}

/**@brief Function for initializing the Advertising functionality.
 */
static void advertising_init(void)
{
    uint32_t      err_code;
    ble_advdata_t advdata;

    // Build advertising data struct to pass into ble_advertising_init().
    memset(&advdata, 0, sizeof(advdata));

    advdata.name_type               = BLE_ADVDATA_FULL_NAME;
    advdata.flags                   = BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE;

    ble_adv_modes_config_t options = {0};
    options.ble_adv_fast_enabled  = BLE_ADV_FAST_ENABLED;
    options.ble_adv_fast_interval = APP_ADV_INTERVAL;
    options.ble_adv_fast_timeout  = APP_ADV_TIMEOUT_IN_SECONDS;

    // Create a scan response packet and include the list of UUIDs
```

```
    ble_advdata_t srdata;
    memset(&srdata, 0, sizeof(srdata));
    srdata.uuids_complete.uuid_cnt = sizeof(m_adv_uuids) / sizeof(m_adv_uuids[0]);
    srdata.uuids_complete.p_uuids = m_adv_uuids;

    err_code = ble_advertising_init(&advdata, &srdata, &options, on_adv_evt, NULL);
    APP_ERROR_CHECK(err_code);
}

/**@brief Function for initializing buttons and leds.
 *
 * @param[out] p_erase_bonds  Will be true if the clear bonding button was pressed to
    ↪ wake the application up.
 */
static void buttons_leds_init(bool * p_erase_bonds)
{
    bsp_event_t startup_event;

    uint32_t err_code = bsp_init(BSP_INIT_LED | BSP_INIT_BUTTONS,
                                 APP_TIMER_TICKS(100, APP_TIMER_PRESCALER),
                                 bsp_event_handler);
    APP_ERROR_CHECK(err_code);

    err_code = bsp_btn_ble_init(NULL, &startup_event);
    APP_ERROR_CHECK(err_code);

    *p_erase_bonds = (startup_event == BSP_EVENT_CLEAR_BONDING_DATA);
}

/**@brief Function for the Power manager.
 */
uint32_t radio_notification_init(uint32_t irq_priority, uint8_t notification_type,
    ↪ uint8_t notification_distance)
{
    uint32_t err_code;

    err_code = sd_nvic_ClearPendingIRQ(SWI1_IRQn);
    if (err_code != NRF_SUCCESS)
    {
        return err_code;
    }

    err_code = sd_nvic_SetPriority(SWI1_IRQn, irq_priority);
    if (err_code != NRF_SUCCESS)
    {
        return err_code;
    }

    err_code = sd_nvic_EnableIRQ(SWI1_IRQn);
    if (err_code != NRF_SUCCESS)
    {
        return err_code;
    }
```

```
    // Configure the event
    return sd_radio_notification_cfg_set(notification_type, notification_distance);
}

void SWI1_IRQHandler(bool radio_evt)
{
    if (radio_evt)
    {
        /*if (led_status == 1) {
                neopixel_set_color_and_show(&m_strip, led_to_enable, red, green, blue);
                led_status = 0;
        }*/

    }
}

static void power_manage(void)
{
    uint32_t err_code = sd_app_evt_wait();
    APP_ERROR_CHECK(err_code);
}

/**@brief Function for application main entry.
 */
int main(void)
{
        //nrf_gpio_cfg_output(24);
    nrf_gpio_cfg_input(29, NRF_GPIO_PIN_NOPULL);        //PGOOD
    nrf_gpio_cfg_output(28);
    nrf_gpio_pin_set(28);               //PWR_ON1
    nrf_gpio_cfg_output(25);
    nrf_gpio_pin_clear(25);             //SUSP
    //nrf_gpio_cfg_input(26, NRF_GPIO_PIN_NOPULL);              //CHRG
    nrf_gpio_cfg_output(27);
    nrf_gpio_pin_clear(27);             //STBY
    nrf_gpio_cfg_output(0);
    nrf_gpio_pin_set(0);                //HPWR
    nrf_gpio_cfg_input(17, NRF_GPIO_PIN_NOPULL);

    NRF_GPIO->PIN_CNF[24] = (GPIO_PIN_CNF_SENSE_Disabled << GPIO_PIN_CNF_SENSE_Pos)
                                    | (GPIO_PIN_CNF_DRIVE_S0S1 << GPIO_PIN_CNF_DRIVE_Pos
    ↪ )
                                    | (GPIO_PIN_CNF_PULL_Disabled <<
    ↪ GPIO_PIN_CNF_PULL_Pos)
                                    | (GPIO_PIN_CNF_INPUT_Disconnect <<
    ↪ GPIO_PIN_CNF_INPUT_Pos)
                                                                        | (
    ↪ GPIO_PIN_CNF_DIR_Output << GPIO_PIN_CNF_DIR_Pos);

    uint32_t err_code;
    bool erase_bonds;

    // Initialize.
    neopixel_init(&m_strip, dig_pin_num, leds_per_strip);
```

```
    timers_init();
    buttons_leds_init(&erase_bonds);
    ble_stack_init();
    device_manager_init(erase_bonds);
    gap_params_init();
    services_init();
    advertising_init();
    conn_params_init();
    init_SLAVE_1();
    // Start execution.
    application_timers_start();
    radio_notification_init(3, NRF_RADIO_NOTIFICATION_TYPE_INT_ON_ACTIVE,
    ↪ NRF_RADIO_NOTIFICATION_DISTANCE_800US);
    err_code = ble_advertising_start(BLE_ADV_MODE_FAST);
    APP_ERROR_CHECK(err_code);
    nrf_gpio_cfg_output(10);
    nrf_gpio_cfg_output(19);
    nrf_gpio_cfg_output(20);
    nrf_gpio_pin_clear(20);
    // Enter main loop.
    for (;;)
    {
        power_manage();
    }
}

/**
 * @}
 */
```

Lihtlitsents lõputöö salvestamiseks ja üldsusele kättesaadavaks tegemiseks ning juhendaja(te) kinnitus lõputöö kaitsmisele lubamise kohta

Mina, _____,
(*autori nimi*)
sünniaeg _____,

1. annan Eesti Maaülikoolile tasuta loa (lihtlitsentsi) enda loodud lõputöö

_____
_____
_____,
(*lõputöö pealkiri*)

mille juhendaja(d) on_____,
(*juhendaja(te) nimi*)

1.1. salvestamiseks säilitamise eesmärgil,
1.2. digiarhiivi DSpace lisamiseks ja
1.3. veebikeskkonnas üldsusele kättesaadavaks tegemiseks

kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile;
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Lõputöö autor _____
(*allkiri*)

Tartu, _____
(*kuupäev*)

Juhendaja(te) kinnitus lõputöö kaitsmisele lubamise kohta

Luban lõputöö kaitsmisele.

_____ _____
(*juhendaja nimi ja allkiri*) (*kuupäev*)

_____ _____
(*juhendaja nimi ja allkiri*) (*kuupäev*)